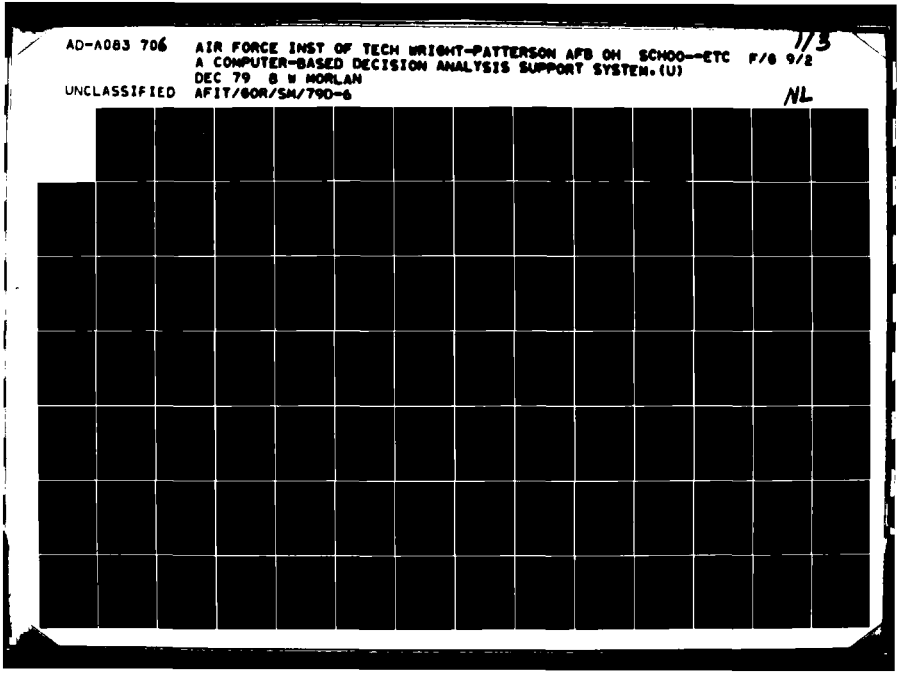
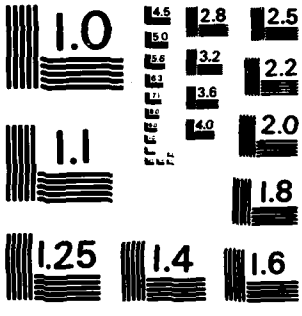

AD-A083 706
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A COMPUTER-BASED DECISION ANALYSIS SUPPORT SYSTEM.(U)
DEC 79 B W MORLAN
UNCLASSIFIED AFIT/GOR/SM/79D-6
1/3
NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

A COMPUTER-BASED DECISION ANALYSIS SUPPORT SYSTEM

THESIS

AFIT/GOR/SM/79D-6

Bruce W. Morlan
Captain USAF

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFIT/GOR/SM/79D-6 — 2. GOVT ACCESSION NO.: AD-A083 706	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: A COMPUTER-BASED DECISION ANALYSIS SUPPORT SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: MS Thesis
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Bruce W. Morlan, Captain, USAF	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: 6 December 1979
	13. NUMBER OF PAGES: 175
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release: distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release IAW AFR 190-17

Joseph P. Hipps, Major, USAF
Director of Information

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computer programs
Decision aids
Decision analysis
Decision making
Decisions
Hierarchies
Mission area analysis
Multiattribute utility

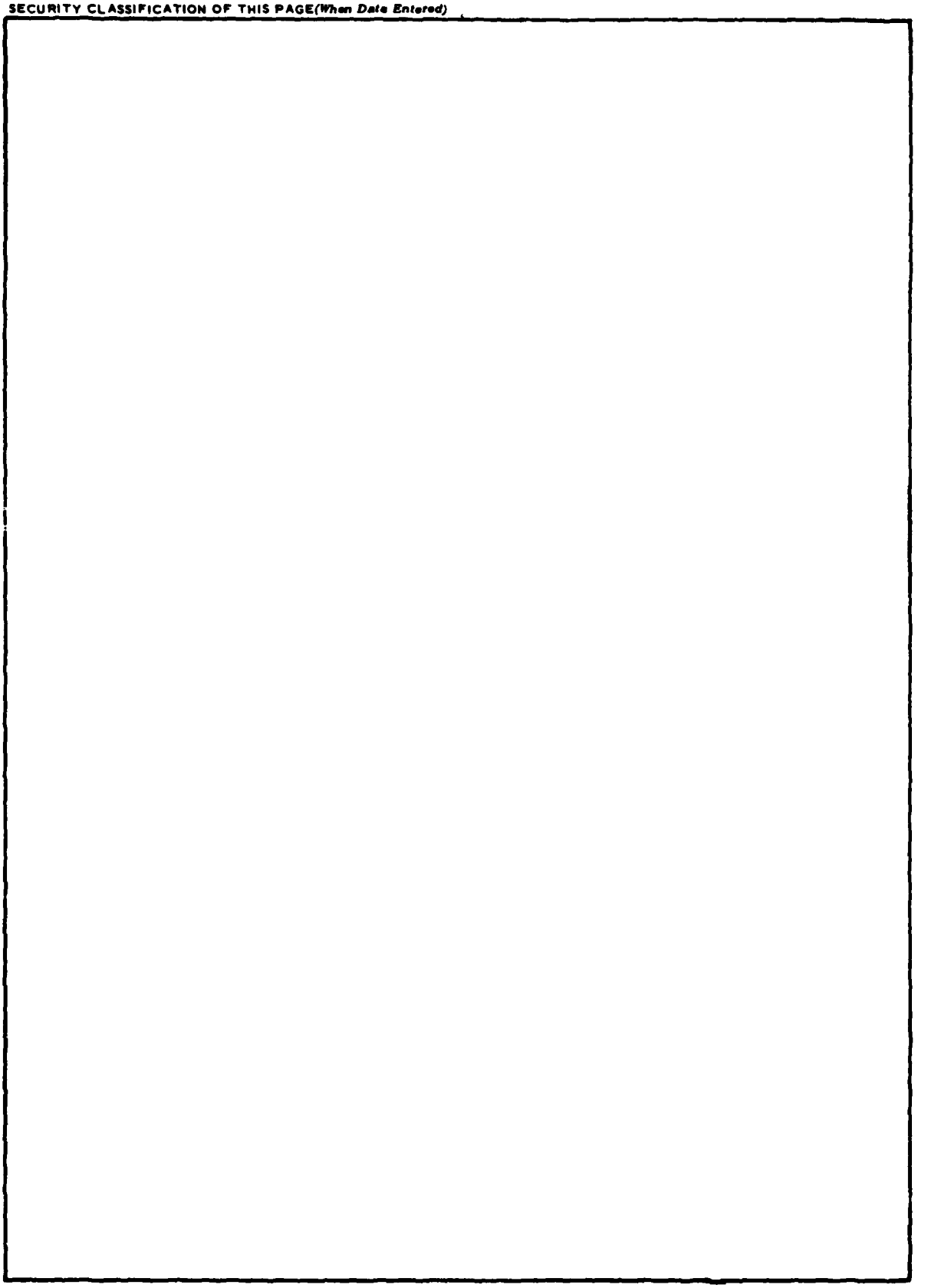
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Decision analysis paradigms often involve hierarchies, and a computer program was created for use as an interactive aid for decision makers and analysts to use in manipulating these hierarchies. A stand alone user's guide, with a highly documented FORTRAN program listing are included. Also discussed are the sensitivity analysis algorithms and the tree manipulating algorithms.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*



AFIT/GOR/SM/79D-6

Thesis

A COMPUTER-BASED DECISION ANALYSIS SUPPORT SYSTEM

by

Bruce W. Morlan
Captain USAF

Prepared in partial
fulfillment of the
requirements for a
Masters Degree

December 1979

School of Engineering
Air Force Institute of Technology
Wright-Patterson Air Force Base
Ohio

1

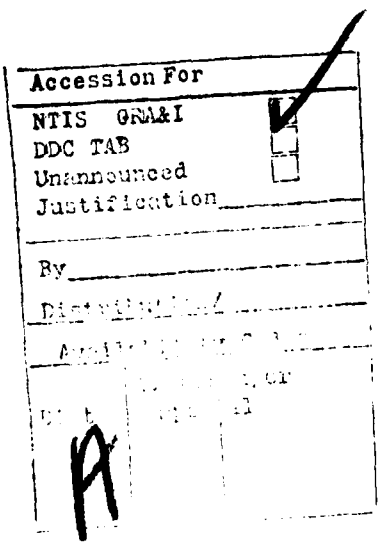
PREFACE

I wish to thank the people at the Defense Advanced Research Projects Agency for their support, comments and suggestions. I especially owe a debt of gratitude to Captain Michael Hayes (USN), who gave me many suggestions and much encouragment.

I also wish to thank my advisor, Colonel Charles Margenthaler, who gave me a balance of freedom and guidance which allowed me to develop this thesis. My thanks also go to Dr. Al Moore, my reader, who gave good advice and made good sense.

My classmates, in a class by themselves, also deserve my thanks. They helped me through the hard times, and ensured that I kept the proper perspectives.

Finally, my thanks to my wife Becky, and my sons Merrick and Jeremy, who gave more than can be told that this thesis could be completed.



ABSTRACT

Decision analysis paradigms often involve hierarchies, and a computer program was created for use as an interactive aid for decision makers and analysts to use in manipulating these hierarchies. A stand alone user's guide, with a highly documented FORTRAN program listing are included. Also discussed are the sensitivity analysis algorithms and the tree manipulating algorithms.

Table of Contents

List of Figures

A COMPUTER-BASED
DECISION ANALYSIS SUPPORT SYSTEM

I. INTRODUCTION

In a world characterized by ever decreasing resources and ever increasing complexity, a decision maker is faced with constantly increasing requirements to make the best decisions. Often the alternatives can include very difficult questions of trade-offs, especially in the Department of Defense, where many trade-offs are between such high value factors as: lives, national security, deterrence, and international peace. The increased complexity, caused in part by increased awareness and in part by decreasing resources, has brought about a need for new, sophisticated methods for organizing information for the decision maker.

Although the basic concept of making decisions as one of choosing between alternatives is easy to understand, the actual process is often very difficult. There are many factors which can make a decision very hard to make. Among these factors are such things as: (1) uncertainty about the future state of the world, (2) incommensurable measures (money, political costs, lives saved, etc.), (3) uncertainty about the set of feasible alternatives, (4) variable, scenario dependent

requirements, and (5) the size of the problem (dimensionality). One model which is useful when making these kinds of difficult decisions is that of decision analysis.

Decision Analysis

> "Decision analysis is an analytical methodology for helping individuals and organizations to make better decisions, principally by structuring the relationships among the various considerations which enter into any decision. A complex decision problem is decomposed into clearly defined components, such as options, uncertainties, and values. It is then structured as a formal and dynamic (emphasis added) decision model ..." (Ref 1:1)

Decision analysis is an attempt to help structure the decision making process. Some familiar forms of the decision analysis model include: (1) the decision tree, which breaks the decision down into components of; (a) decisions among simple alternatives, and (b) chance (non-choice) events, (2) the regret/value matrix, which measures choices among alternatives against the possible states of nature, (3) multiattribute utility (MAU) theory, which breaks the payoff down into components, (4) mission area analysis (MAA) which categorizes the state of the world by missions, conditions, criteria and systems (Ref 2:A-4).

Each of these forms can be mapped onto a basic decision analysis paradigm which contains a weighted hierarchical network. This hierarchical form is convenient both from the aspect of describing the problem and from the aspect of manipulating the information to get results which can aid the decision maker.

Statement of the Problem

It is an unfortunate fact that many problems are so complex that the decision makers or analysts find themselves overwhelmed by the sheer

size and number of interrelationships. This can result in analyses which are incorrect, or incomplete.

The computer has, in some respects, aggravated the problem by providing even more information for the decision maker to process. Recent changes in the perception of how the computer can be used have led to many exciting new decision aiding systems. Among these new systems are Probabilistic Inference Programs (PIP), new simulation languages, and Decision Support Systems (Ref 17 and Ref 14).

The computer provides a convenient means of storing and manipulating the decision analysis paradigm structure and its associated data. Because the computer can perform thousands of calculations per second without error and can store the large amounts of data associated with some decision analysis models, it represents an extremely useful tool for the analyst or decision maker. Unfortunately, the human-machine interface is often inefficient. Even though the computer can handle large amounts of data, the human element is incapable of dealing effectively with more than just a few cues at one time. Proper interfacing is an essential element in recapturing that information. Among the methods which exploit the computer's capabilities are: (1) sensitivity analysis, (2) automatic searching for important pieces of data, (3) good graphical displays, and (4) an interactive dialogue between man and machine which can provide immediate feedback of results in a real-time interaction (Ref 7).

The core problem, then, is to provide the highly interactive, real-time, decision aids which decision makers and analysts can use together to resolve problems. Programs are needed which provide the types of outputs which are needed for making good decisions, from the

types of inputs which users are willing to input.

Objective

The objectives of this thesis were derived from the initial research into the topic area. In addition to an extensive literature review, discussions were conducted with various practitioners of decision analysis, as well as with people working on the theory of hierarchical decision models. The objectives, as derived from this process, were to create a seed program which could

1) be used interactively in nonspecific hierarchical decision analysis

2) provide sensitivity analysis which could be used in an interactive dialog

3) demonstrate some of the display formats which could be used to get information to the user

4) act as a foundation upon which to build a decision analysis package

5) demonstrate some of the algorithms which can be used for manipulating hierarchical information

6) be documented sufficiently for other prospective users to implement and modify

This thesis consisted partly of creating a program which could aid in manipulating weighted hierarchies of the type which have been previously discussed. The system which was created was called a "Decision Analysis Support System (DASS)." The DASS carried forward program concepts which were developed (in part) for the Defense Advanced Research Projects Agency (DARPA). The original work was performed for DARPA by Decisions and Designs, Inc. (DDI). DDI originally developed a series of programs which aided in decision analysis. These programs are quite restricted, however, in that they are written in a relatively rare

language (IBM APL). This language is available on the IBM 5100 series computers, and on some large mainframe computers. It is a language which is not easily translated to other languages.

The Program. As stated, there were other codes available to perform some of the DASS functions. These codes suffered from several restrictions. Some of these restrictions included: (1) programs coded in APL, a language which is not commonly available, (2) non-interactive programs, (3) programs which were poorly structured for sensitivity analyses, or (4) programs which could not display the information satisfactorily. These problems were some of the those kept in mind during the design and development of the DASS.

We felt, and DARPA agreed, that a rewriting of the basic package was needed to overcome these problems. Research was conducted into the types of programs available, and into the types of program structures which could be used to create a new decision analysis package.

Although DARPA made available a copy of EVAL (an interactive program for manipulating hierarchies) as written in APL by DDI, it was decided that the time which would be spent learning the highly symbolic language (APL) would be better spent on developing a new program. Several advantages accrued from using this approach. First, it meant that the programs were well understood, making it easy to improve or add to the programs. Second, it meant that the programs could be well structured, by using a top-down structured programming approach. This makes it easier for others to use and modify the programs at a later date. Third, this made the package more internally consistent.

The Computer. A collateral objective of this thesis was to demonstrate the feasibility of using a small, inexpensive computer

system in decision analyses. The Defense Advanced Research Projects Agency (DARPA) accepted a proposal asking for funding for such a system (in particular, an Apple II home computer was requested). The original DASS was written in FORTRAN and a second version was written in Applesoft II BASIC.

This second version demonstrates nicely the possibility of using very inexpensive systems in decision analysis. In addition to the low cost, it was felt that such a system could be better utilized in interacting with the decision maker(s) involved. With such a system, it is possible to do real-time sensitivities, graphs and model modifications in response to the typical barrage of "what if" questions.

Documentation. Proper documentation can spell the difference between a "living" and a "dead" program. Too many beautifully conceived and well written programs are lost because the original author has left. Too many programs are created for one problem, then recreated later by another user because the first program was unreadable and indecipherable. DASS was to be very well documented, both for users and for programmers. Documentation for the DASS is contained in this thesis and its appendices. Among the appendices are: (1) a user's manual for DASS, (2) a highly commented FORTRAN program listing, and (3) a discussion of the more useful algorithms, as well as information about the meanings of the variables, and how they are used in the program.

II. STATE OF THE DECISION SCIENCES AND DEVELOPMENTAL METHODOLOGY

As the capabilties of computers have grown, the decision sciences have always been at the leading edge of applications. Until recently the primary uses of computers in the field have been in simulations, massive data manipulation and other non-interactive uses. A new concept which has been developing recently is the concept of interactive decision aids. Among the new applications which have appeared are such interactive, real-time tools as: (1) PIP, Probabilistic Inference Processors, (2) MIS, Management Information Systems, (3) DSS, Decision Support Systems, and (4) decision analysis systems (Ref 17:117,Ref 14).

Probabilistic Inference Processing (PIP) systems are aids for use in determining probabilities of events. They are designed to help an analyst capture the interrelationships of available present world data, and to use that information to project into either the future (what will happen) or into the undetermined present (what is happening). The systems work using Bayesian probability theory, subjective judgments and (sometimes) Delphi techniques. The reader is referred to references 14 and 5 for discussions on the current state of research and development of PIP's.

Management Information Systems (MIS) are systems which are designed to aid management in sifting through data relevent to a problem. They are primarily information processors, and usually do not provide interpretation or integration of information.

The Decision Support Systems (DSS) are among the latest concepts in computer aids to decision making. P. G. Keen identifies two classes of interactive computer programs which are in use by managers: data base management systems (MIS) and finacial planning (modeling, analysis, etc.). A term was needed to describe this new concept of the proper relationship of managers/decision makers to computers. The term coined was Decision Support Systems (DSS).

> "A decision support system is a computer-based system (say, a data base management system or a set of finacial models) which is used personally on an ongoing basis by managers and their immediate staffs in direct support of managerial activities -- that is, decisions." (Ref 17:117)

The concept of DSS encompasses many of the traditional systems, and indirectly includes the decision analytic programs which are being developed.

Decision analysis, as previously stated, is an approach to problem solving which attempts to structure the problem in terms of smaller subunits which can then be dealt with on an individual basis. Although decision analysis has been in use for some time, it, like many other of the decision sciences methods, had not completely come to grips with the rapidly expanding capabilities of the computer industry.

Recent work done by Decisions and Designs, Inc. (DDI) for the Defense Advanced Research Projects Agency had demonstrated the possibilities inherent in on-line, real-time decision analysis, using models which could: (1) perform hierarchically based probability assessments (program OPINT), (2) perform multiattribute, linear additive value function analysis (program EVAL), and (3) work with standard decision tree structures (program DECISION). Unfortunately, the

aforementioned programs, while operational, are not readily available, primarily due to the fact that they had originally come about as a result of DDI's internal requirements, and as such they suffered from several shortcomings. Among the shortcomings were: (1) the language used (APL), while powerful, is not available as widely as some other languages, e.g. FORTRAN and BASIC, and if available, is not as well known, (2) the programs were not well documented, making additions and modifications difficult, and (3) they were limited in the size of the problem which they could handle.

On the other hand, it appeared that there were certain desirable properties held in common by most of the usable decision aiding programs which were available. Each of the systems had at least two major factors in common: (1) they were real-time programs which provided immediate answers to the questions which they were designed to answer, and (2) they were designed to be usable without a high degree of computer expertise. We felt that if the project was to be successful, the program, in addition to meeting these user requirements, would have to meet the types of requirements that prospective programmers would look for in any program which could act as a basis for further work. This led to certain problems. First, to make a program non-specific enough to allow easy application to a wide variety of problems, it was necessary to keep the interactive prompts as general as possible. This requirement conflicts with the requirement that the program be heavily user oriented. A trade-off was made, resulting in a satisfactory compromise position. The second problem was that the program needed to be very modular in structure, which results in some duplication of coding in an effort to maintain clarity, which is essential if the

program is to be modifiable by any other user/programmers.

The result of this effort was the Decision Analysis Support System. The Decision Analysis Support System (DASS) is a program concept which was designed to aid decision makers and analysts who must work with hierarchical decision analytic models. The program is designed to meet several requirements: (1) provide a usable decision aid which has real-time interactions with the user, (2) provide real-time sensitivity analyses, (3) demonstrate some of the display formats which can be of use to a decision maker, (4) provide a program which is capable of being moved from computer to computer with a minimum of reprogramming effort (through the use of a common language and good documentation), and (5) demonstrate some of the useful algorithms which can be used to process tree structures effectively.

The need was seen for a program package which could act as a seed for a system which could do for decision analysis what PIP does for Bayesian inference, what MIS does for managing information and what DSS does for supporting decisions. This program is such a seed, and to be viable it was necessary for it to: (1) be written in a form which could be modified easily, (2) be written understandably, (3) be implementable on most systems on which analysts work, and (4) be usable in its initial form, lest it never be used or improved. Of primary importance was writing a program which could be <u>modified</u> by interested users. It is almost axiomatic of programs, just as of life, that change is inevitable, and that when change ceases, so too does life. The usual cycle of a new computer program is that the original creator will use the program profusely, and that as the organization changes the program is handed down to each new monitor with a loss of information, until the

program is either lost or enshrined.

It was these types of problems which it was hoped could be avoided. Essentially, every effort was made: to make the program modifiable, use a top-down structured programming approach; to make the program readable, highly documented code was used; to make the program implementable on most systems, FORTRAN was used; and to ensure the use of the program, a wide dissemination was sought. The documentation for this program is found in the appendices to this thesis.

III. THE STRUCTURE OF THE DECISION ANALYTIC MODELS

Hierarchies are found at the heart of many forms of decision analysis. They are used as a means of decomposing a single focal objective into managable subobjectives. The essential characteristics of a hierarchy are: (1) a vertical structure and (2) the characteristics of a component (node) are dependent on the characteristics of the lower level components below it (Ref 2:A-1). The most familiar form of hierarchy is probably the family tree, from which many terms are borrowed for use in discussing hierarchies. The glossary contains definitions for many of these terms.

Often, as one progresses down through hierarchy, the nature of the subobjective changes (e.g. objectives become subobjectives, which become goals, tasks or attributes). The meanings of the components and relationships in the context of various decision analysis paradigms is discussed in the following text.

Decision Trees. Probably the most common form of hierarchy encountered in decision analysis is that used in decision trees. This hierarchy is constructed of nodes which represent: (1) decisions among a finite number of options, and (2) random or externally controlled events among a finite set of possibilities. Brown (Ref 9) provides an interesting discussion of this use of hierarchies, and points up some serious shortcomings of the decision tree as it is usually taught. This form was not addressed in creating the DASS, although the program could

easily be modified to handle this application of hierarchies.

Mission Area Analysis. Mission Area Analysis (MAA) represents a very generalized use of the hierarchy for decision analysis. In MAA the various levels of the hierarchy represent differing aspects of the problem. In Air Force applications of MAA, the hierarchy may be broken down into four categories: (1) missions, at this level in the hierarchy, the tasks to be performed to accomplish the mission are indicated, (2) conditions/constraints, these levels contain the operational constraints such as weather, geography and threats, (3) operational criteria, these levels provide the measures against which to assess the abilities of the systems (reliability, maintainability , sustainability, availability, etc.), and (4) systems, this is the lowest level of the hierarchy, containing the systems and resources which might be applied to accomplish the stated mission (Ref 2:A-4) (see figure 3.1).

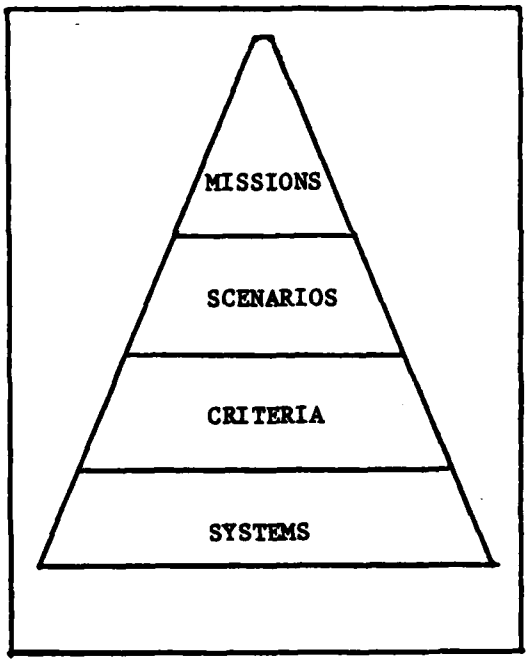


Fig 3.1 Air Force MAA

Multiattribute Utility Applications. In the context of Multiattribute Utility (MAU) applications, an unmeasurable attribute (such as "the best job") is decomposed into several subattributes, each of which is itself then decomposed, until a level is reached which can be measured (hopefully objectively, but often subjectively). For

example, "the best job" as an attribute for comparing job options might be decomposed into elements of "pay", "location" and "job satisfaction" (see figure 3.2) (Ref 18:422-426). These attributes could then be further decomposed, until the attributes are sufficiently isolated to be measured for value assessment.

Management by Objectives. In the context of Management by Objectives (MBO), the organizational objectives are defined by subobjectives, goals or tasks. An organizational objective of maintaining a leadership position in the relevant industrial sector might be decomposed into subobjectives of marketing, research and development, and community involvement Then each of these could be further decomposed into more detailed objectives, until goals can be applied (e.g. at a lower level, the goal of maintaining a 58% share of the market).

In each of these examples, the lower levels represent a closer, more detailed look at the information contained in the higher levels of the hierarchy. The primary objective of this decomposition is to clarify the nature of the main objective, and to provide a means of dividing a problem into components which are (hopefully) more easily handled.

Multiattribute Utility Theory

Multiattribute Utility (MAU) theory is a structured approach to decision making which attempts to overcome the "information overload" of a multidimensional outcome by allowing the decision maker to evaluate simpler subsets of attributes. The overall value of the various alternatives is calculated as a function of the values of the decomposed subset values.

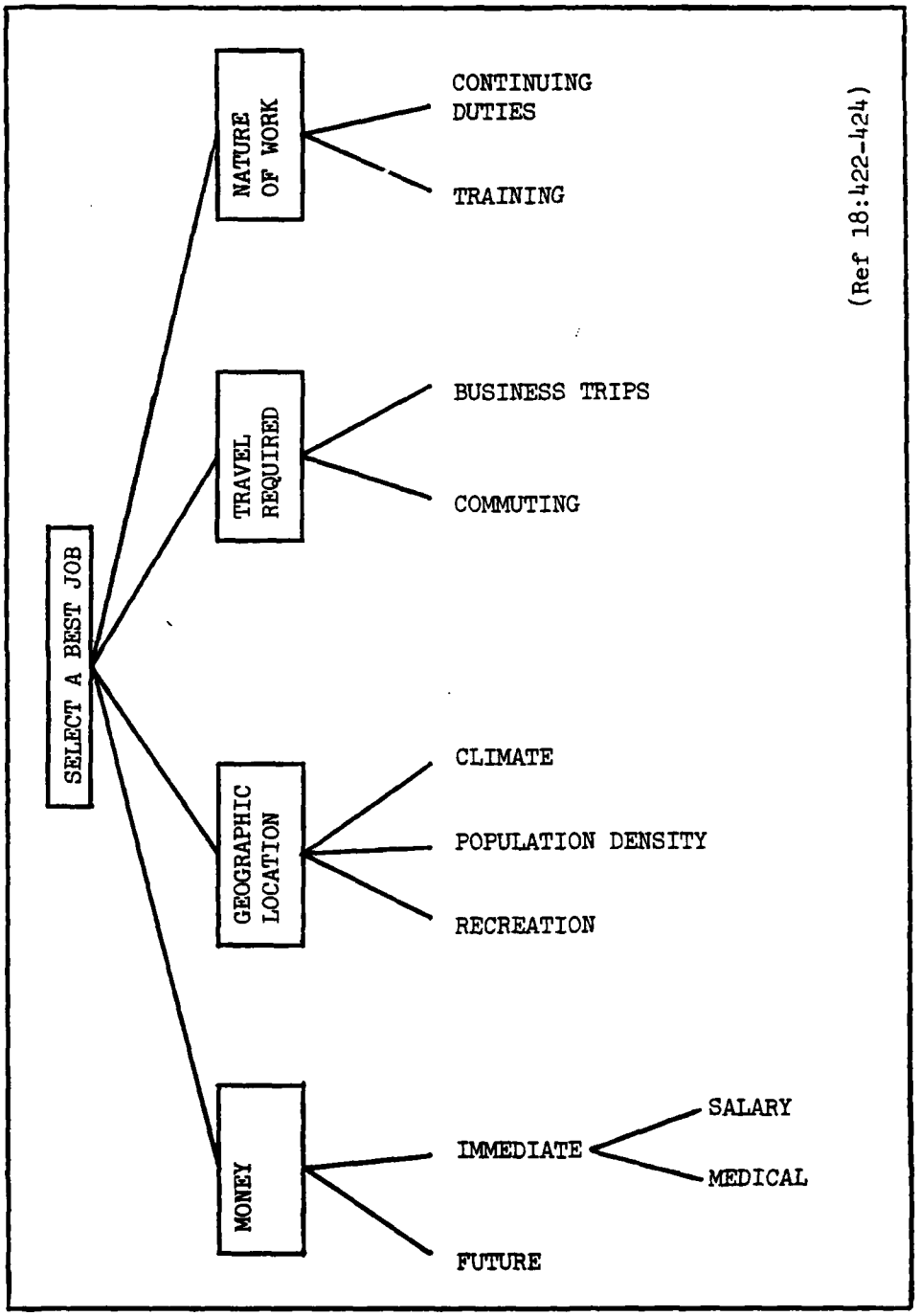


Fig 3.2 A Multiattribute Value Function

The preference between two outcomes is fairly clear when that outcome is measured as a scalar ($10 is preferred to -$15), when the outcome is a vector the choices become more difficult. For example, which is the preferred outcome, (1) to spend $5M per year to save an expected 36 lives per year, and have a 0.6 probability of being reelected, or (2) to spend $20M per year to save an expected 53 lives per year, and have a 0.34 probability of being reelected? The choice between $10 and -$15 is trivial, but the choice on the vector of dollars, lives and reelection is far from trivial, yet it is of the type that a politician choosing a city wide ambulance service might be expected to make (see footnote).

Desirable Properties of Attribute Sets. In creating the decomposed set of attributes, certain properties are desirable. The chosen attribute set should be complete, operational, decomposable, and nonredundant (this list of properties and the following discussion thereof is from Keeney and Raiffa (Ref 18:50-52)).

Completeness. A set of attributes is complete if it captures the preferences of the decision maker. For example, an analyst who analyzes the preceding ambulance example using only the attributes of money and lives might not understand why the politician failed to follow the recommendations given. Without considering the attribute of reelectability , the set of attributes is not complete.

Operational. The concept of being operational is similar to that of completeness. A set of attributes is operational if it captures the measures of the outcomes. If completeness is thought of as a function

---

Note - we assume that all outcomes are deterministic, not probabilistic unless stated otherwise. A complete discussion of decisions under uncertainty is not undertaken here.

of the decision maker, then an operational set is "complete" with respect to the problem being considered.

Decomposable. The property of being decomposable is that it is possible to treat subsets of the attribute set independently. It is this property which allows the use of the hierarchical model to any advantage.

Nonredundent and Minimal. The set of attributes should not be redundent, and should not be any larger than necessary. Realizing that the determination of values for attributes (discussed later) is not an exact science, a cutoff on attributes must be made before the set becomes unmanagabley large. Just as in regression analysis, one must capture most of the information but an attempt to capture it all will rapidly lead to an unusable set of attributes.

Value Functions

Having determined a set of attributes, an ordering over the space of possible outcomes must be provided. The question which needs answering is "what is the value of a particular outcome (state)"? Although this is sometimes easy, such as when the outcome is measured solely in dollars, there are many more times when the outcome is in some vector of outcomes such as money, health and political gain. An ordering is needed so that, given two outcomes, $\bar{x}_1$ and $\bar{x}_2$, it is possible to determine which is the preferred. Returning to the ambulance problem, the relevent choices result in outcome vectors of

($5M, 36 lives saved, p(reelection)=.6)

and ($20M, 53 lives saved, p(reelection)=.34)

These points are not ordered, and there is no direct means of

determining which is preferred by the decision maker.

The solution usually used is to reduce the outcomes of decisions to values. A value function which maps from the space of all possible outcomes into an ordered scalar space (usually the reals) is a means of providing an ordering which can be used to determine which of two vectors of attribute levels is preferred. A value function is defined as:

$$\text{Value} = \text{Function}(x_1, x_2, \ldots, x_n)$$

where

x_i are the various measures of merit (attributes)

This value function can then be used to rank order differing outcome vectors. The usual definitions are (x bar ($\bar{x}$) denotes a vector):

if $V(\bar{x}) > V(\bar{y})$ then $\bar{x}$ is preferred to $\bar{y}$ $(\bar{x} \succ \bar{y})$

if $V(\bar{x}) = V(\bar{y})$ then $\bar{x}$ is indifferent to $\bar{y}$ $(\bar{x} \sim \bar{y})$

By using a value function to collapse the information in a multidimensional outcome, a well-ordered set has been produced, as for every $\bar{x}$ and $\bar{y}$, one and only one of the following holds. Either

$\bar{x}$ is preferred to $\bar{y}$ $(\bar{x} \succ \bar{y})$

$\bar{x}$ is indifferent to $\bar{y}$ $(\bar{x} \sim \bar{y})$

or $\bar{x}$ is less preferred than $\bar{y}$ $(\bar{x} \prec \bar{y})$ (ordered)

further

$\bar{x}_{worst} \prec \bar{y}$ for every $\bar{y}$ (well-ordered)

The value function is defined over the space of possible outcome vectors, and provides an ordering to the possible outcomes. Then, if the outcomes are deterministic (based on only the decision) then one need only maximize the value, the decision which does so is the best decision.

Although the MAU function can take on many forms, the linear additive form is often used, primarily because it is so easy to use and understand. The basic form of the linear additive MAU function is

$$V(\bar{x}) = \sum_{i=1,n}((w_i)x(v_i(x_i)))$$

where

v_i is the value function of x_i alone

v_i(worst value of x_i) = 0

v_i(best value of x_i) = 1.0 (or other constant)

$w_i > 0$ weighting factor for the i-th value function

$\sum_{i=1,n}(w_i) = 1.0$

This form, and the methods for evaluating v_i and w_i are discussed extensively by Keeney and Raiffa (Ref 18). It is this form which is most easily applied to decision analysis.

Assumptions

There are certain assumptions which merit close attention and discussion if the linear additive model is to be used.

Independence. The linear form of the MAU function assumes value independence. This means that the value of x_i is not dependent on the levels of the other attributes. This is a crucial assumption, and one which may not be justifiable (Ref 14). For example, the value of

having a given number of aircraft is strongly dependent on the amount of support (e.g. maintenence) which is available.

Constant Marginal Rate of Substitution. The marginal rate of substitution of x_i for x_j is

$$MRS = \frac{w_j}{w_i}$$

which is a constant. This says that the marginal value (in terms of x_i) which will be given up for an increase in x_j is not dependent on the level of x_j. This is contradictory to most economic models of behavior (see figure 3.3).

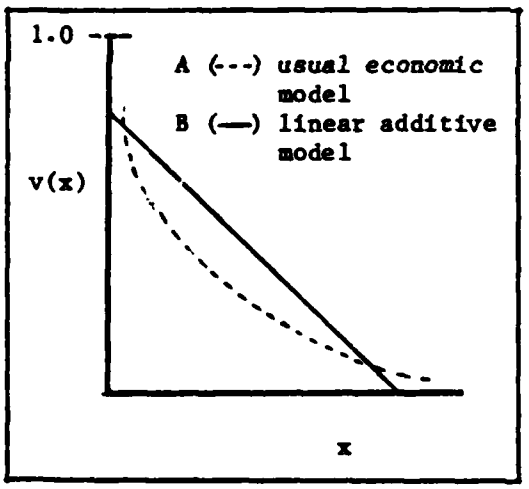


Fig 3.3 Value Functions

These assumptions serve to weaken the case for using linear additive models, yet they are still widely used. The usual rationale for using it is the simplicity. Additionally, the problems are not as significant in the center regions of the range of outcomes, and it is not a bad approximation to use the linear approximations there.

The Hierarchical Paradigm

We now turn to a discussion of the basic hierarchical paradigm. Essentially, a hierarchy is a tree, where tree is used in the sense of an undirected, connected network without cycles (Ref 21:45). The

structure of trees is a mathematically defined concept, and as such there are well defined terms which can be used in a discussion of trees. Most of these terms are defined in the glossary, and they will not be repeated here.

The topmost node of the tree is the root, and all other nodes are descended from it. All nodes have certain characteristics which are common to all the uses discussed here, although the name of that characteristic may change with the application. The key fact is that although the name changes, the mathematical properties of the characteristic is unchanged.

Values. The first aspect of a node which is of interest is the value associated with the node. In an abstract sense, the value at a terminal node is some measure of the degree of benefit (or regret) which is associated with that node. For example, if the hierarchy is modeling a multiattribute utility function, then the value is the scaled value of an option or decision for that bottom level attribute. On the other hand, in a Mission Area Analysis, the value would be the given systems contribution to the selected mission, given the scenario and a particular operational criterion. In one Air Force application of MAA, the value used is capability and the ultimate measure is a transformation of capability to "need", where

$$\text{need} = (\text{weight}) \times (1-\text{capability})$$

(Ref 2:C-3)

Consider the comparative value of two jobs (JOB1 and JOB2) as shown in figure 3.2 (this structure was adapted from one developed (more completely) by J. R. Miller which was discussed in

reference 18:423-425).

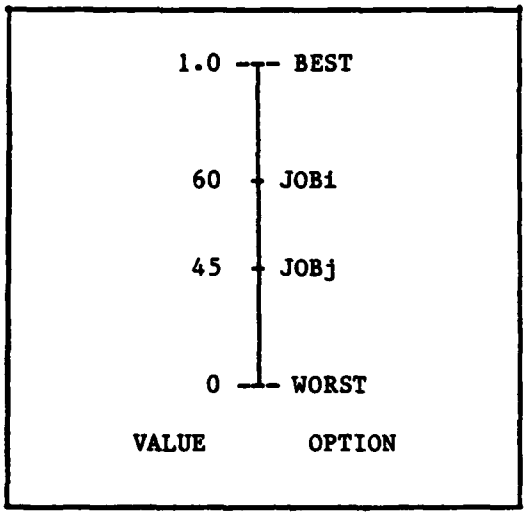


Fig 3.4 Subjective Values

When the attribute of money-immediate is used as a measure, it is relatively simple to assign a quantitative value. When the attribute of geographic-climate is considered, a subjective judgment is required (figure 3.4). This subjective judgment must be made directly from the system to the value of the system.

There are two contexts for discussion of the values associated with a tree: (1) as they are found at the terminal (input) nodes: and (2) as they are found at internal (branching) nodes.

Values at Terminal Nodes. The values which are associated with a tree are input at the terminal nodes and calculated from these values for all other nodes (this is discussed later in more detail). The input values are scaled to some common value (usually 1.0 , 100.0 , or -100.0). This common value would represent the maximum return from some idealized optimal combination of circumstances and choice. If the input weights are multiplied times the cumulative weight at the terminal node, the result is a measure of that node's contribution to the total score of the option or system being considered. The ratio of this value to the total value for the system would be an indicator of the importance of that node's power in discriminating between options.

Thus, a low cumulative weight coupled with a high score might be as important as a high cumulative weight and low value.

Values at Internal Nodes. The values at the internal nodes are calculated from those input at the terminal nodes. This is done by using a recursive definition, which states that the value at a internal node is:

$$V_{internal} = \sum_{i=children}((V_i)x(W_i))$$

where

$V_{terminal\ node}$ = input value

W_i = weight relative to siblings

Although this definition is workable, there is a second, equivalent definition which proved to be more workable from a programming standpoint. This second definition is that

$$V_{node} = \sum_{I}((V_i)x(CUMWT_i))$$

where

I = the set of descendent terminal nodes

V_i = the value of the system at the i-th terminal node

i = index over I

$CUMWT_i$ = cumulative weight of the i-th node

Local Values. The local value associated with a node ($V_{L,i}$) has the cumulative weight of that node factored out.

$$V_{L,i} = V_i \ / \ CUMWT_i$$

This allows a more direct comparison of different nodes and their contributions to the whole. This also makes it easier to compare different option values at a node.

Global Values. The global values associated with a node represent the raw sum of the values of that node's descendents, and is not often used, although it could be divided by the total system score to get a fraction of the contribution to the total system score from a given node.

Weights. One characteristic of a node is its weight. All nodes except the root have a weight (although the weight of the root is often defined to be 1.0 , this is just for computational ease). The weight of a node can also be discussed in two contexts.

The first context for discussing the weights is as they relate to the weights of the siblings. In this context, the weight of a node is its contribution to the parent node. A usual restriction is that the weights of the children of a node sum to 1.0 .

$$\sum_{i=\text{children}} (w_i) = 1.0$$

Given compliance with this restriction, the weight of the node is its fraction of contribution to the parent. Using figure 3.5 as an example, the relative weights of nodes AA , AB , and AC are 0.5 , 0.2 , and 0.3 respectively. The relative weights of ABA and ABB are 0.5 and 0.5 .

The second context for discussion of weights is as they relate to the entire tree. In this context, the weight becomes the node's contribution to the root node, rather than the parent node. This weight is often referred to as the cumulative weight, a name which derives from

the fact that the cumulative weight is the cumulative product of the weights of all the nodes between the root and the selected node.

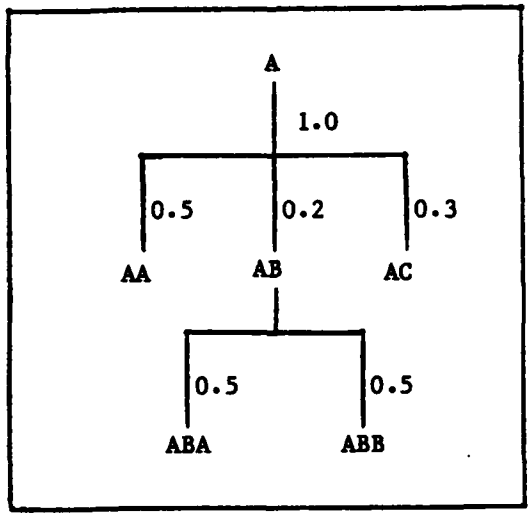


Fig 3.5 A Weighted Tree

Again using figure 3.5, the cumulative weight of node AA is 0.5 (1.0 x 0.5) and the cumulative weight of ABA is 0.1 (0.2 x 0.5). One consequence of these definitions is that the sum of the cumulative weights over any set of nodes which cuts across the entire tree without duplicating a cut will also be 1.0 . In the example, the set of nodes {AA,ABA,ABB,AC} cuts across the tree, and the sum of the cumulative weights is 1.0 = (0.5 + 0.1 + 0.1 + 0.3). A tree which is constructed this way is referred to as a normalized tree.

Determining Values

Once the structure of the hierarchy has been constructed, it becomes necessary to determine the values and weights to be assigned to the nodes. Determining the values is discussed first as the determination of the weights is controlled by the form of the values used.

The values carry differing meanings depending on the model under consideration. In MBO, for example, the values represent the degree of completion of a task. When combined (multiplicatively) with the cumulative weight, the value becomes a direct measure of the

contribution of that terminal node to the overall objective. In the case of MAU models, the values are a measure of the value of the system being considered as measured against some other system (possibly an ideal). Referring to our earlier discussion, the ideal or comparative system would be given a value of 1.0 . The subject system is then measured against the comparative system. Note that even if the measure is quantifiable , such as missions per day, the values may not be simple proportions to the comparative system, as the value function used need not be linear. Thus, while the direct measure is missions per day, the value measure might be a function such as

$$v(\text{missions per day}) \propto \text{SQRT}(\#\ \text{missions per day})$$

This particular value function is shown in figure 3.7.

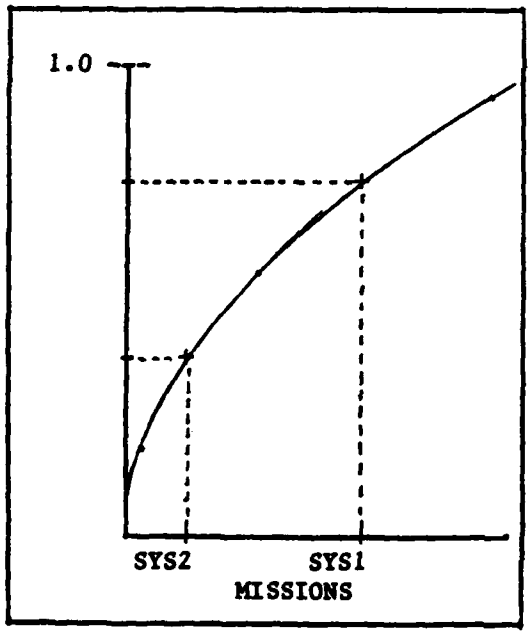


Fig 3.6 A Value Function

Note that while SYS1 will fly four missions per day while SYS2 will fly only one, the value of SYS1 is not four times the value of SYS2.

If the form of the value function is unknown, or if it is not possible to quantify the attribute, then a relative scaling method can be used. Reconsider the earlier example of choosing a job.

In determining the values of the various options (JOB1,...,JOBn) at the terminal node geography-climate , there is no quantifiable measure of

value, unless one wanted to try using measures such as average temperature, rainfall etc. Ultimately, however, the decision maker must make a direct value judgment, whether of a quantitatively or qualitatively measured attribute.

The method which seems most usable is that discussed by Keeney and Raiffa (Ref 18). They propose that all the options be scaled to a common scale from 0.0 to 1.0 where value(worst system)=0.0 and value(best system)=1.0 . The decision maker must then determine the fraction describing the position of all other systems on such a scale. The position assigned is a subjective evaluation of the worth of the system relative to the ideal-worst difference. Although this seems to be fairly arbitrary, the facet of the hierarchy which allows it to work is the weight which is assigned to each node.

Determining Weights

If we have the relative values of the systems, then all we need is some means of combining them into a scalar (to get the ordering relation). In the linear additive model, the form of the model also adds an interpretation to the weights which can be used in determining the weights to be used.

Before discussing this interpretation, a short discussion of the fundamental meaning of the weights is in order. The nature of the weights of the nodes is a function of the paradigm which is being used. Mathematically, the weights are treated the same, regardless of the exact meaning assigned to them.

In the case of a multiattribute utility function, these weights reflect several factors. Remember that the basic form of the linear MAU model is

$$U(\bar{x}) = \sum_i ((w_i)x(U_i(x_i)))$$

where

$U(\bar{x})$ is a utility function over a multidimensional attribute space

w_i is a weighting factor for the i-th dimension

$U_i(x_i)$ is the i-th utility function (which is assumed to be independent of the other w_j, $j \neq i$)

$\sum_i(w_i) = 1.0$

$0.0 < w_i < 1.0$ for every i

$U_i(\text{worst } x_i) = 0.0$ for every i

$U_i(\text{best } x_i) = 1.0$ for every i

(Ref 13:458 and Ref 18:118-119)

We must determine what the decision maker's values are for these importance weights. There are at least three direct means of accomplishing this: (1) direct assessment at each terminal node of the hierarchy, (2) assess each different branching node (taking advantage of symmetries), then take the product through the tree, (3) assess each branching node, conditional upon the nodes above it (Ref 20:42-44).

In the first method, direct assessment, we find that a medium number of assessments must be made, but that we have essentially lost the advantage of using the hierarchical structure for the model. Direct assessment of the importance weights at each terminal node may be difficult, and the consistency of the weights is difficult to ensure.

Assessing each level separately, then using the same weights for

each member of the family of nodes which have the same downlink structure has the advantage of reducing the number of assessments to a minimum (relative to the other methods), but it has the significant disadvantage of making an almost certainly invalid assumption about the independence of the nodes. The possible invalidity of this assumption is clearest in the case of a Mission Area Analysis model, where the weight of a branch is certainly dependent upon the up-level branches. In this case, the assumption contradicts one of the primary reasons used to justify the use of MAA, which is that it allows for the variance caused by varying the circumstances.

The third method, that of assessing each node conditional upon the preceding nodes, offers the most accurate approach (in terms of theoretically capturing the most information). This method makes the best use of the hierarchical structure, using it to help reduce the dimensionality of the judgments required, while creating a framework within which to make those judgments. With this method, each branch is given a weight relative to its siblings, which is determined within the full context of the nodes which are above it in the tree structure. These weights are then combined multiplicatively, as in the second method. This method has the extreme disadvantage of requiring the largest number of assessments, as each node must be assessed individually.

An Interpretation to Aid in Weighting. Earlier it was suggested that the technique of scaling the values of the systems to an ideal system would aid in determining the proper weights. What is provided is a means of crosschecking for consistency, and a means of starting to determine the weights when the subjective evaluation is very difficult.

Consider the ambulance problem previously discussed, and assume that the final structure is as shown in figure 3.7. Consider what it means to say that the weight of the value of life is 0.5 when the weight of the value of money is also 0.5 . This means that the decision maker is indifferent between the outcomes

$\bar{x}_1$ = (100 lives, \$50M spent,?)

with a value of $v(\bar{x}_1) = 1.0x0.5 + 0.0x0.5 = 0.5$

and $\bar{x}_2$ = (0 lives, 0 spent,?)

with a value of $v(\bar{x}_2) = 0x0.5 + 1.0x0.5 = 0.5$

This demonstrates the fact that the decision maker has costed out the value of 100 lives as being equal to the value of \$50M .

Keeney and Raiffa propose the following method for using this to determine the proper weights (Ref 18). Consider the worst case vector

x_o = (0 lives,\$50M, p(reelect)=0)

If only one of the attribute dimensions can be increased to its maximum, while the others remain the same, then which would the decision maker choose? Assume that the choice is to go to the vector

(100,\$50M,0)

What has been learned? From this choice we know that $w_1 > w_2$ and that $w_1 > w_3$. Continuing with the other alternatives, we can eventually get an ordering for the w_i, for example

$$0 < w_3 < w_2 < w_1 < 1.0$$

This is more information than was initially available, and is a place from which to start the process of refining the weights down to their

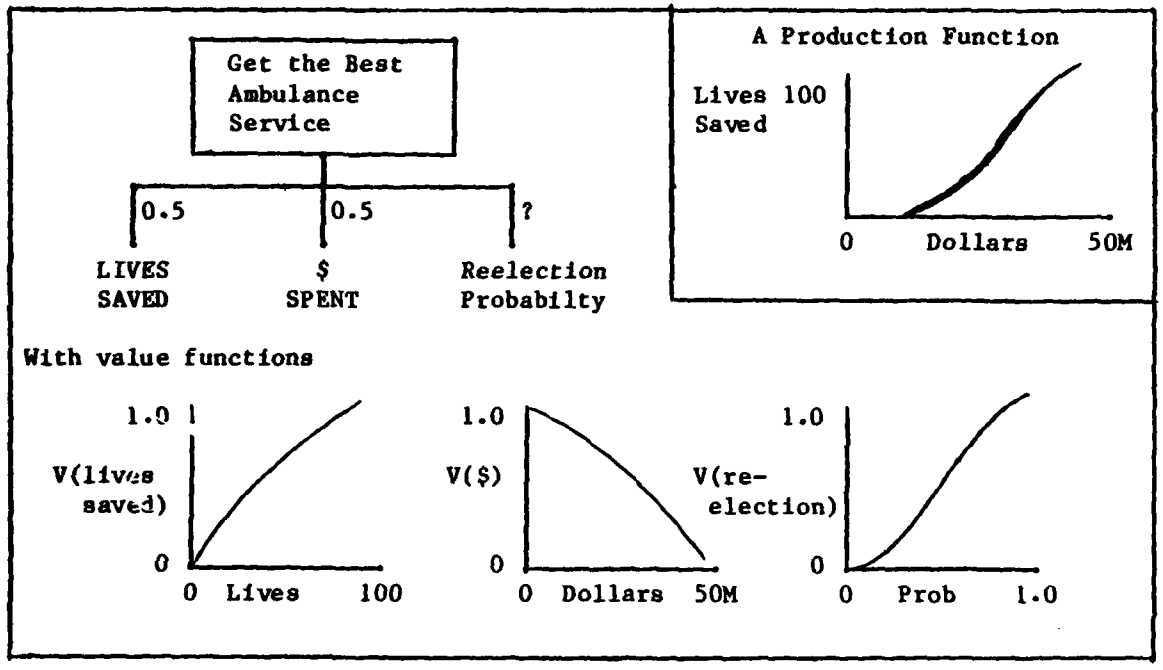


Fig 3.7 The Ambulance Tree

appropriate values. We can now test various other outcome pairs for indifference, until we are confident that we know the weights sufficiently well to use them in the analysis. The reader is referred to reference 18 for an excellent discussion of the topic of weight estimation.

IV. A SAMPLE APPLICATION

The following is a brief illustrative example of the use of a Decision Analysis Support System (DASS) in a multiattribute value decision. This example is for demonstration purposes only.

The Problem

An Air Force officer (say Colonel) is faced with a seemingly simple decision concerning which of three available aircraft types to stage to a forward base. The Colonel doesn't know which of the three would best fill the needs of the situation, and all three seem to be viable. The decision must be made fairly quickly, and the staff is called in to aid in the decision. (See footnote)

After some discussion, it was determined that the the Colonel's value structure for an aircraft, given the current scenario, was as shown in figure 4.1. Although the Colonel considered only three aircraft types to be viable, it was decided to leave the possibility open for more to be added at a later time. Thus, the values elicited were scaled against an ideal system, and not against one of the three.

After determining the structure of the hierarchy, and the values of the terminal nodes, the weights were determined for the various subobjectives. All of the weights as determined are shown in figure

---

Note-this example was structured to parallel an actual decision (made in a business context) which was discussed in reference 17:118. It uses a final resolution which is similar to the real-world decision discussed

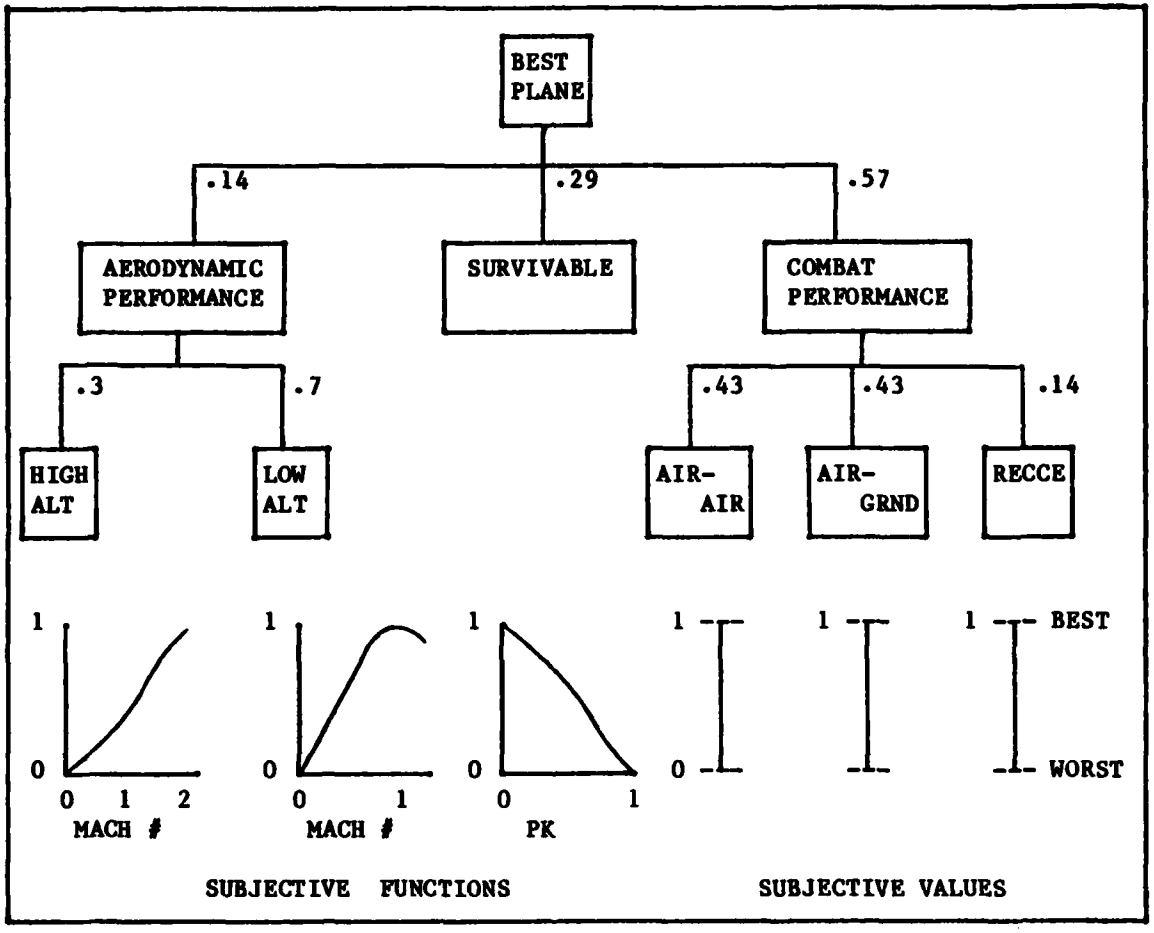


Fig 4.1. A Simple Hierarchical Problem

4.1. (See footnote)

After the preliminary model had been constructed, it was entered into a Decision Analysis Support System. The program calculated the final, composite values shown in figure 4.2. This information in this figure represents the consolidation of all the information in the tree. Thus, for example, the value of 55.04 , as found in the TOTAL row of the F-15 column represents the subjective value of the F-15 in this example. This can be compared with 100.00 (the ideal) to get an idea of the performance of the F-15, but a better comparison is against the composite values for the other systems.

BEST PLANE- FACTOR	WT	F-4	F-15	F-111	CUMWT
1)AERO	(.14)	34.50	31.00	47.50	.14
2)SURVIVABLE*	(.29)	50.00	80.00	40.00	.29
3)COMBAT	(.57)	50.00	48.57	34.29	.59
TOTAL		47.79	55.04	37.81	1.00

Fig 4.2 Table of Final Values

When compared, the value of the F-15 is found to be higher than the values for either of the other two systems being considered. As this problem was structured with preference being indicated by a higher value, it is apparent that the preferred system is the F-15. When shown this result, the decision maker was not completely satisfied. The F-15 was not the airframe that a visceral response would have indicated as being the best. A second look at figure 4.2 shows that the F-15 rates

Note-the eliciting of values and weights for this type of model is a subject for another thesis, and is not discussed here. Also, these numbers were generated by a non-pilot friend of the author, and should not be compared with actual values.

first in only one catagory, SURVIVABLE, although it is a close second in the area of COMBAT. The question which immediately comes to mind is the importance of survivability for this assignment.

This question, rephrased as "how sensitive is the result to the weight on survivability?", was put to the analysts. A sensitivity analysis, performed real-time, was run on the cumulative weight of the SURVIVABLE branch. The decision maker's initial estimate of the weight was .29 , and the weight sensitivity analysis was run over the entire possible range of 0.0-1.0 . The results were summarized by the program as shown in figure 4.3.

```
 WEIGHT  F-4     F-15    F-111
 0.00    46.90*  45.06   36.93
  .10    47.21   48.55*  37.24
  .20    47.52   52.05*  37.54
  .30    47.83   55.54*  37.85
  .40    48.14   59.03*  38.16
  .50    48.45   62.53*  38.46
  .60    48.76   66.02*  38.77
  .70    49.07   69.52*  39.08
  .80    49.38   73.01*  39.39
  .90    49.69   76.51*  39.69
 1.00    50.00   80.00*  40.00

          0--------20--------40--------60--------80--------100
WT =0.00 +         +         C +  B     +         +         +
WT = .10 +         +         C+   B     +         +         +
WT = .20 +         +         C+   A B   +         +         +
WT = .30 +         +         C+   A   B +         +         +
WT = .40 +         +         C+   A     B         +         +
WT = .50 +         +         C+   A     +B        +         +
WT = .60 +         +         C+   A     +  B      +         +
WT = .70 +         +         C    A     +    B    +         +
WT = .80 +         +         C    A     +      B  +         +
WT = .90 +         +         C    A     +       B +         +
WT =1.00 +         +         C    A     +         B         +
          0--------20--------40--------60--------80--------100
```

Fig 4.3 Sensitivity to Weight of SURVIVABLE

It was immediately apparent that the F-15 was better as long as the weight of SURVIVABLE never dropped below about 0.2.

The next question raised concerned the weight of the AIR-GRND capabilties. What would happen if the weight of AIR-GRND were raised somewhat? Again, an immediate sensitivity analysis could be run, and the results are shown in figure 4.4.

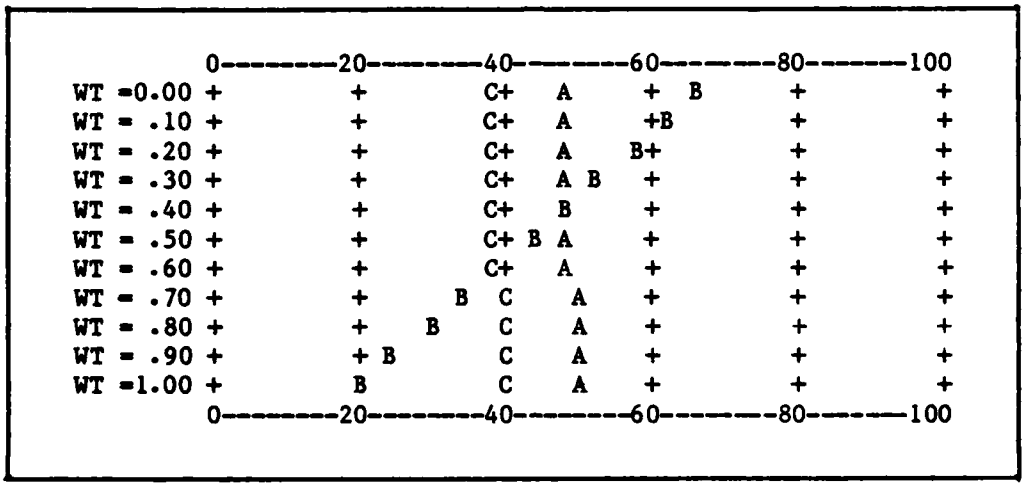


Fig 4.4 Sensitivity to Weight of AIR-GRND

Now a source of concern is apparent. The F-4 becomes better than the F-15 if the weight of the value of AIR-GRND action increases by just a small amount. Noting that the risk involved in using the F-4 was less than the risk involved in using the F-15 (the F-4 curve is quite flat through the entire relevant region), the option chosen was to go with the F-4.

Summary

This simple example demonstrates the advantages which can be gained through the use of on-line decision analysis aiding programs. The decision maker is able to immediately look at areas which may be cause for concern, and can look for the sensitive values, those values

and weights which can easily change the best answer (such as the AIR-GRND considerations in the example). The slope of the sensitivity line, while probably invalid far from the estimated value (per earlier discussions), does give an idea of the importance of a particular estimate.

V. CONCLUSIONS AND RECOMMENDATIONS

Although there has been much interest in decision analysis in recent years, and some development of theoretic means of using the tools of decision analysis, there is still much work to be done in the area of interactive, on-line, real-time decision aiding computer programs. In the process of the research for this thesis, much interesting work was found which demonstrated the potential for such systems.

The contribution to the field of decision analysis provided through this thesis is the Decision Analysis Support System (DASS) around which other analyst/programmers can build. Creation of such a system was a primary goal of the research effort. Imbedded in the philosophy of its design is an independent user's manual and a well documented program. As far as can be determined, the research provides the first explicit description of an algorithm for sensitivity analysis. Readers will find in the documentation concise descriptions of the hierarchical tree manipulating algorithms and of the variables used in the program. These descriptions can be of great use if additions or modifications are to be made. Of significance is the stand alone user's manual and programmer's guide.

Areas for Further Work

The DASS program is a first phased effort. A number of additions are viewed as being desirable to enhance the capabilities of the

program. For example, the number of caveats (in the form of cautions in the user's manual) is viewed as excessive for an interactive, user-oriented system. Replacement of many of these caveats with internal checks and balances is certainly desirable. Many of the required program changes are simple changes which were left out due to time constraints.

The Applesoft Plus (BASIC) microcomputer version of the DASS is incomplete. It suffers from some of same problems as the FORTRAN version. It does demonstrate the feasibility of using small inexpensive systems for decision analysis problems, although their computational speed may make it difficult to use them for large problems. The BASIC program listing and user's guide are to be published under a separate cover.

Recommended Extensions to This Thesis

There are several immediate projects which are recommended as extensions to this thesis. They are:

* Enhanced sensitivity analysis. The basic method used for sensitivity analyses (described in the programmers guide) could easily be extended to such sensitivities as
  (a) sensitivities on the values selected
  (b) simultaneous sensitivities at two nodes (weights and/or values)
  (c) Risk Assessment. A form of automatic sensitivity searching, which yields an idea of the riskiness of choosing a particular system. This is something which could detect the hazard found in the example in Chapter IV, where the apparent best system was also excessively sensitive to the weights.

* Make the model more specific to a particular function, say Mission Area Analysis. Improve the interactive dialog within the context of that function.

* Application of the models, programs, and processes to several different real world problems.

* Study the effectiveness of various displays.

Less direct extensions include

* Measure actual value or utility functions. Investigate the extent of the error introduced by assuming that the functions are linear in form (to what degree is the sensitivity analysis affected by this assumption?).

* What is the affect of node dependence? Is it realistic to assume node independence?

Other Areas for Research

In addition to the specific possibilities mentioned, it appears that there are several broad areas which need further work. First and foremost, there is a requirement to continue with the development of the models and the computer programs necessary for on-line decision analysis. These programs will not appear without the dedicated efforts of persons well trained in both the decision sciences and the computer sciences.

Second, there is work to be done on the questions of measurement and validity of the subjective judgments used in the decision making process. This is a three-fold question. First, are the simplifying assumptions made (such as linearity) close enough to modeling reality to give good results? If not, then what models would be better? Second, what is the effect of measurement error on the validity of the output from the models? Remember, decision makers are human, and that makes them stochastic events when it comes to making measurements of their thoughts. It may be that sensitivity analysis is enough to offset this, but many statistical measurement studies need to be done in this area. Third, what is the proper role of the results of decision analysis? Already known is the fact that most decision makers do not want a

machine that will make the decisions for them. This is not even desirable. What must be addressed is the extent to which this occurs defacto. Quoting one student who was exposed to a decision analysis exercise

> "My major problem with the process is the credibility which the numbers assume once selected. Weights are subjective, yet once selected they begin to control." (Ref 17:1)

This problem is one which is a question of ethics for an analyst, and as such is easily handled. The concientious analyst will always make certain that the decision maker understands the limitations of any analyses which are performed. However, if the analyst is removed totally or partially from the loop, through the use of highly interactive programs, one runs the risk of the decision maker failing to apply the correct degree of uncertainty to the results. Further, as the decision maker is brought more closely into the actual process of analysis, the decision maker may find it harder to keep the correct perspectives.

A third area for further research is that of the proper display formats for the results of decision analysis. Here we are not discussing just the most effective combination of tables, graphs and words, although that is part of the question. We are also talking about the deeper question as to what constitutes valid displays, displays that capture both the useful information and the caveats which should go with the results.

In conclusion, the decision analysis process delves into the most complex areas, where difficult trade-offs are made, where values are brought out for all to see, where there are no clean edges, but only

fuzzy indistinctions. The problems which are confronted with decision analysis are those which have no "school" solution, or they wouldn't have been brought to this level of analysis. At the same time, these questions are among the most interesting, stimulating, and challenging to work on, and this author can only promise the analyst entering this arena an interesting area in which to work.

Bibliography

1. _____. Decisions and Designs, Incorporated (DDI), Business flyer, McLean, VA:DDI (Feb 1979?).

2. _____. USAF Procedures Guide to Mission Area Analysis, (Jan 1979).

3. _____. Preliminary Users Manual for EVAL Program, McLean, VA:DDI for DARPA (1979). (not published)

4. Amey, Dorothy, Phillip H. Feuerwerger and Roy M. Gulick. Documentation of Decision-Aiding Software: OPINT. McLean, VA:DDI for DARPA (April 1979).

5. Barclay, Scott. Interactive Graphic Aids for Bayesian Hierarchical Inference. Technical Report:Rome Air Development Center, Air Force Logistics Command (Dec 1976) (RADC-TR-76-308).

6. Barclay, Scott and Cameron R. Peterson. Multi-Attribute Utility Models for Negotiations. Technical Report:DDI for DARPA (March 1976) (DDI TR-76-1).

7. Barclay, Scott, et al. Handbook for Decision Analysis. McLean, VA:DDI for DARPA (Sep 1977) (DDC# AD A049221).

8. Brandt, Thomas C., Howard E. Bethel and Wallace B. Frank, Jr. "Mission Area Analysis Allocation for Air Force R&D," Defense Systems Management Review, Vol II:77-92 (Spring 1979).

9. Brown, Rex V. "Heresy in Decision Analysis: Modeling Subsequent Acts Without Rollback," Decision Sciences Vol 9:543-554 (1978).

10. Buckley, Shiela R. National War College Memorandum for: Commandant, 13 Jun 77, Subject:Computer-Assisted Political-Military Simulation.

11. Buede, Dennis M, et al. Applications of Decision Analysis to the US Army Affordability Study. Technical Report:DDI, McLean VA (Dec 1978) (DDI TR-78-10-72).

12. Buede, Dennis M. The Use of the Reconnaissance/Surveillance Mission Area Analysis by a Program Office. Final Report:DDI for HQ TAC/XP-ALPHA (Nov 1977) (DDI PR-77-15-75).

13. Fisher,Gregory W. "Utility Models for Multiple Objective Decisions: Do They Accurately Represent Human Preferences?," Education Vol unk:451-479 (1979).

14. Fisher, Gregory W., Ward Edwards and Clinton W. Kelly, III. Decision Theoretic Aids for Inference, Evaluation, and Decision Making: A Review of Research and Experience. McLean, VA:DDI (Feb 1978) (DDI Technical Report TR$78-1-30, DDC AD053962).

15. Goodman, S. E. and S. T. Hedetniemi. Introduction to the Design and Analysis of Algorithms. New York:McGraw-Hill Book Co. (1977).

16. Hays, Michael L. and Michael F. O'Connor. "Relating Promised Performance to Military Worth: An Evaluating Mechanism," Defense Management Journal:36-46 (Oct 1977).

17. Keen, P. G. and G. R. Wagner. "DSS: An Executive Mind-Support System," Datamation, Vol25:117-122 (Nov 1979).

18. Keeney, Ralph L. and Howard Raiffa. Decisions With Multiple Objectives: Preference and Value Tradeoffs. New York:John Wiley & Sons (1976).

19. Kelly, Clinton W. III, and Cameron R. Peterson. Decision Theory Research. Technical Report:DDI (Sept 1975) (DDI DT/TR-75-5).

20. O'Connor, Michael F., and Ward Edwards. On Using Scenarios in the Evaluation of Complex Alternatives. Technical Report:DDI (1976).

21. Pflaltz, John L. Computer Data Structures. New York:McGraw-Hill (1977).

22. Radford, K. J. Managerial Decision Making. Reston VA:Reston Publishing (1975).

23. Raiffa, Howard. Decision Analysis (Introductory Lectures on Choices Under Uncertainty). Reading MA:Addison-Wesley (1968).

24. Ulvila, Jacob W., Rex V. Brown and Karle S. Packard. "A Case in On-Line Decision Analysis for Product Planning," Decision Sciences, Vol8:598-615 (1977).

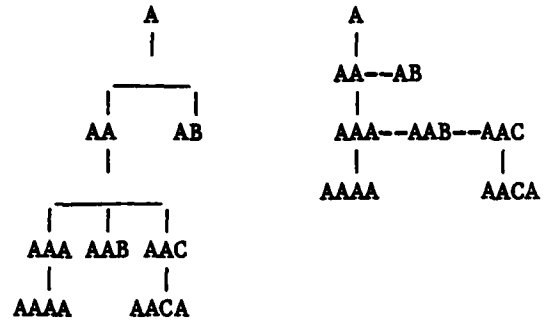
25. Winkler, Robert L. and William L. Hays. Statistics (Probability, Inference, and Decision). New York:Holt, Rinehart and Winston (1975).

APPENDIX A

Glossary

Glossary

Note - the glossary will refer to the following example tree hierarchy to demonstrate some of the concepts defined.



(tree structure) (data structure)

The Sample Hierarchy

backlink - this is a data structure concept. The node which precedes a node in the data structure is backlinked to that node. In the example, node(AA) is the backlink to node(AB) and node(A) is the backlink to node(AA).

branching node - (also internal node) is a node which has at least one descendent

breadth-first search - A breadth-first search moves first to the siblings of a node, then to the descendents. A breadth-first search of the example would visit the nodes in the order

A AA AB AAA AAB AAC AAAA AACA

This is the order used when searching for a match to an input NRN. This is also the path used when creating a new structure using the SPAn mode to enter a tree.

cell (or data cell) - refers to the block of data associated with a node. DASS uses cells to store the

pointers and data for each node. The pointers stored are the backlink pointer, the downlink pointer, and the crosslink pointer. The data stored is the node digit, the weight (relative to siblings), the cumulative weight, the system values (input or calculated), and the user's rationales for the input weights/values.

children - this is a concept of hierarchies. The children of a node are those nodes which are one level down from that node. In the example, nodes(AA,AB) are children of node(A) and nodes(AAA,AAB,AAC) are children of node(AA)

composite value - this is a hierarchical concept. The composite value for a node is the sum of the weighted values of the nodes below it. The composite value for a bottom node is a user supplied value.

crosslink - this is a data structure concept. The first sibling node which is added after a node is that node's crosslink node (crosslinked to that node). In the example, node(AB) is the crosslink to node(AA) and node(AAC) is the crosslink to node(AAB)

cumulative weight - this is the weight of a node, relative to the root node. This is a measure of the contribution of the node to the root. It is equal to the product of the weight of the node (normalized, relative to its siblings) and the cumulative weight of the parent. Thus, CW(AAAA)=CW(AAA)xW(AAAA)=W(AA)xW(AAA)xW(AAAA) where CW(*) is the cumulative weight and W(*) is the relative weight.

data node - a node which has no descendents is a data node

data structure concept - a concept which relates to how the program stores the tree structure in core (arrays) and which is not directly related to the hierarchical structure itself

descendent - a node which comes after a given node in the hierarchy is a descendent of the given node

depth-first search - a synthesis of data and logic structure concepts. A depth-first search traverses a tree by:
(1) adding new levels first (if possible)
(2) then visiting the crosslink nodes
A depth-first traversal of the example would visit the nodes in the order
A-AA-AAA-AAAA-AAB-AAC-AB

node digit - the NRN is a vector which is made up of node digits. For each level down, an additional digit must be added to identify a node. (Note-the node digit may be more than than a single digit, in the sense of digits as numerals)

node reference number (NRN) - the vector which points the path through the tree from the top down to a particular node. Each node has a unique NRN

normalized tree - this is a hierarchical concept. A tree is normalized if the sum of the weights of its bottommost nodes is 1.0 . This, by the nature of the weights, implies that any set of nodes which completely cuts across the tree will also form a set of nodes with the sum of their weights equal to 1.0 .

parent node - this is a concept of hierarchies. The node directly in line one level above a node is the parent node to that node. In the example, node(A) is the parent of nodes(AA,AB) and node(AA) is the parent of nodes(AAA,AAB)

psuedo-array - this is a data structure concept. A psuedo-array is a means of storing information which is later accessed as if it were in an array, but using the psuedo-array name actually accesses a function which returns the information requested

root - this is a concept of hierarchies. The root is the

topmost node in the hierarchy.

sibling - this is a concept of hierarchies. Those nodes which have the same parent are siblings. In the example, nodes(AA,AB) are siblings and nodes(AAA,AAB,AAC) are siblings.

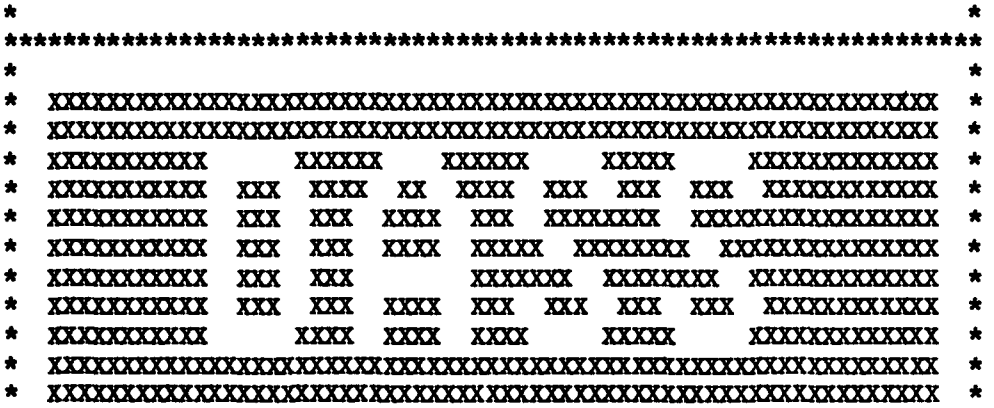
weight (or relative weight) - this is the weight of a node, relative to its siblings. It can be measured either as a fraction of the parent nodes weight, or or as a ratio to the siblings. DASS always normalizes this weight internally such that the weights of a family sum to 1.0 . (See also cumulative weight)

APPENDIX B

User's Manual for

DASS - A Decision Analysis Support System

USER'S MANUAL FOR



A DECISION ANALYSIS SUPPORT SYSTEM

This user's manual was prepared by Captain Bruce W. Morlan as part of a Masters Thesis for the Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1979

Table of Contents

List of Figures

OVERVIEW OF DASS

The Decision Analysis Support System is an interactive program for aiding a decision maker or analyst in processing the hierarchical paradigms which are encountered in decision analyses. Essentially, DASS can manipulate the data associated with a weighted and valued tree structure. The program is not particular about the meanings of the weights or values, as long as certain criteria are met, and it may be used for multiattribute utility models (linear additive), mission area analyses and management by objectives.

CRITERIA

DASS will properly handle most linear additive functions of the type which are typically referred to as weighted trees. Throughout this user's manual, it is assumed that the user is familiar with decision analysis and the terms used therein. The user is referred to the thesis which was written in conjuction with the DASS, especially chapter III, "DECISION MODELS", and to the bibliography for sources of information on decision analysis and the decision analytic paradigms.

Program Capabilities

The Decision Analysis Support System (DASS) is a program concept which was designed to aid decision makers and analysts who must work with hierarchical decision analytic models. The program is designed to meet several requirements: (1) provide a usable decision aid which has real-time interactions with the user, (2) provide real-time sensitivity analyses, (3) demonstrate some of the display formats which can be of use to a decision maker, (4) provide a program which is capable of being moved from computer to computer with a minimum of reprogramming effort (through the use of a common language and good documentation), and (5) demonstrate some of the useful algorithms which can be used to process tree structures effectively.

As the capabilties of computers have grown, the decision sciences have always been at the leading edge of applications uses. Until recently the primary use of computers in the field have been in simulations, massive data manipulation and other uses which use the computer in a non-interactive mode. A new concept which has been developing recently is the concept of interactive decision aids. Among the new applications which have appeared are such interactive, real-time tools as: (1) PIP, Probabilistic Inference Processors, (2) MIS, Management Information Systems, (3) DSS, Decision Support Systems, (4) decision analysis systems (Ref 17:117,Ref 14). Each of these systems has at least two major factors in common: (1) they are

real-time programs which provide immediate answers to the questions which they are designed to answer, and (2) they are designed to be usable without a high degree of computer expertise. These programs do not just "appear", the only means of acquiring these advanced capabilities is for people with backgrounds in both decision sciences and computer programming to expend the effort required to create the systems.

This program was created as part of a thesis which set out to create the basis for a Decision Analysis Support System. The need was seen for a program package which could act as a seed for a system which could do for decision analysis what PIP does for Bayesian inference, what MIS does for managing information and what DSS does for supporting decisions. This program is such a seed, and to be viable it was necessary for this program to: (1) be written in a form which could be modified easily, (2) be written understandabley, (3) be implementable on most systems on which analysts work, and (4) be usable in its initial form, lest it never be used or improved. Of primary importance was writing a program which could be <u>modified</u> by interested users. It is almost axiomatic of programs, just as of life, that change is inevitable, and that when change ceases, so too does life. The usual cycle of a new computer program is that the original creator will use the program profusely, and that as the organization changes the program is handed down to each new monitor with a loss of information, until the program is either lost or enshrined.

It was these types of problems which it was hoped could be avoided. Essentially every effort was made to make the program modifiable by using a top-down structured programming approach; to make

the program readable, highly documented code was used; to make the program implementable on most systems, FORTRAN was used; and to ensure the use of the program, a wide dissemination was sought.

I. INPUT

The term "input" refers to all data which is put into the program. Input comes from two sources: (1) user's interactive responses to program questions (prompts), and (2) previously entered and stored data from disk files. When DASS requires any input from the user it will write a prompting message. There are two types of prompting message: (1) those which end with a question mark, and (2) those which are followed by lines which consist of only a question mark. Any line which does not fall into one of these categories is not expecting a user response.

Program Control

The program is controlled by user responses to questions or by user selection from a menu. Selections are made in one of three ways: (1) using a three character mnemonic (major options), (2) using a number to select from a table of options, and (3) by answering a yes/no question with a 'Y' for yes and an 'N' for no (user may enter extraneous characters in this case, i.e. 'YES' for yes).

Each major option is discussed briefly in the following program flow section concurrent with a simple example. Complete discussions are found in the OPTIONS section.

Conventions

The use of underlined capitalized letters was adopted to show the source of the three character mnemonics used for the major options (i.e.

REView for the review option). All computer input and output is capitalized, except for output variables (i.e. PROMPT-2: CURRENT CUMULATIVE WEIGHT IS cumwt MINIMUM CUMWT (0-1) IS?", the value "cumwt" is a variable, and as such will change, the capitalized portions are fixed). User inputs are set off by delimiters, either square brackets ([,]) or single quotes (').

II. PROGRAM FLOW

The following discussion of the use of DASS is built around the hierarchy shown in figure 1. In the interest of keeping the discussion brief, only a multiattribute value problem was considered. This was deemed sufficient as the Mission Area Analysis and Management by Objectives cases are quite similar in structure.

Consider figure 1, which shows a simple hierarchy for a problem formulation. The question to be answered is, "which of the available systems (F-4, F-15, and F-111) should we deploy to a forward base"? The attribute which we wish to determine the value of is "the best plane", which has been described in terms of the remainder of the tree. At the point when we begin to use DASS, we have a structure (the tree), and we have tentative weights and values.

The first step is to enter the tree structure into the program files. Upon starting, DASS will determine if the file number 1 already contains a tree. If it does not, DASS will automatically take the user through the steps required to create a new tree structure. The logic path used is shown in figure 2, and can be manually selected if the user wishes to overide an existing file.

The first option selected is the TiTLe option, which allows the user to enter a title for the structure about to be entered. This title is later used as a header for certain of the print options. It is recommended that the user provide such information as name, date,

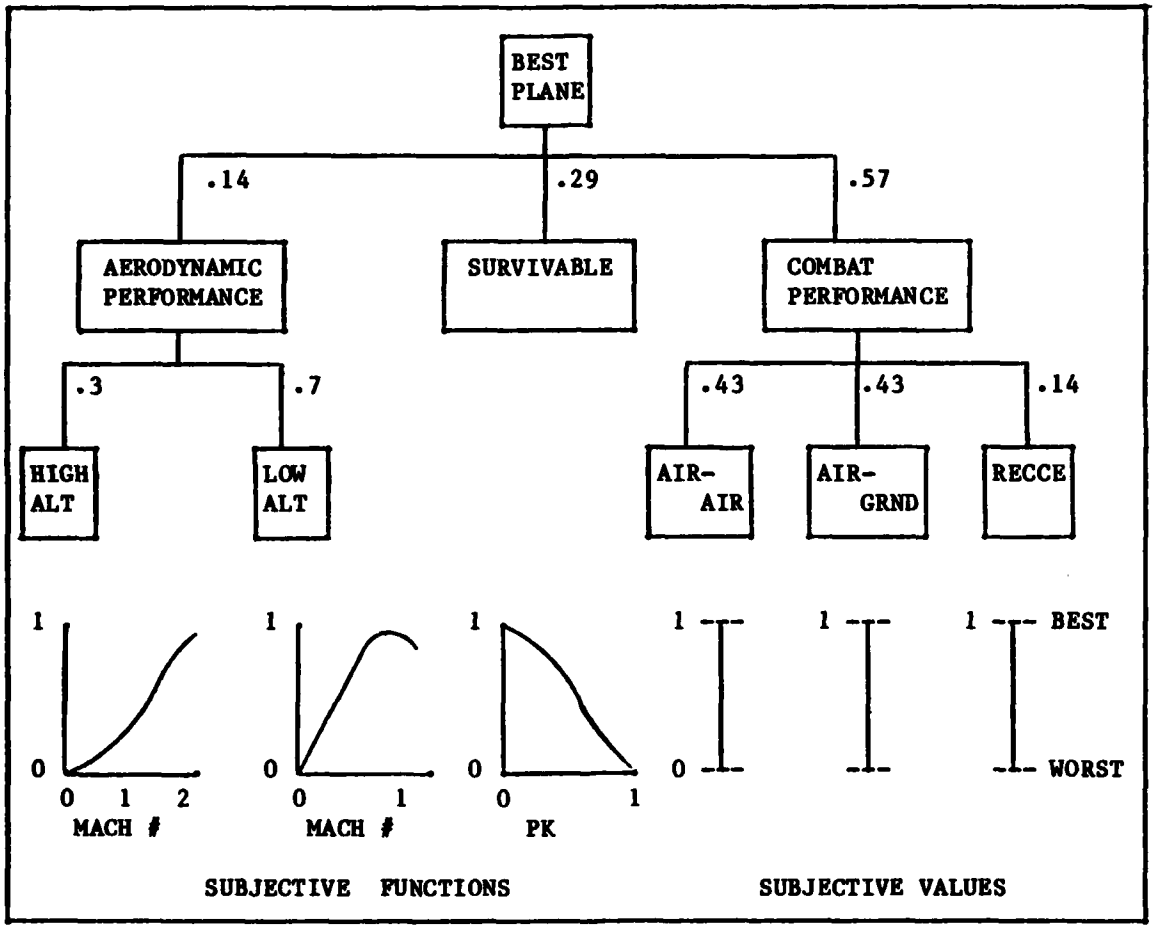


Fig 1. A Simple Hierarchical Problem

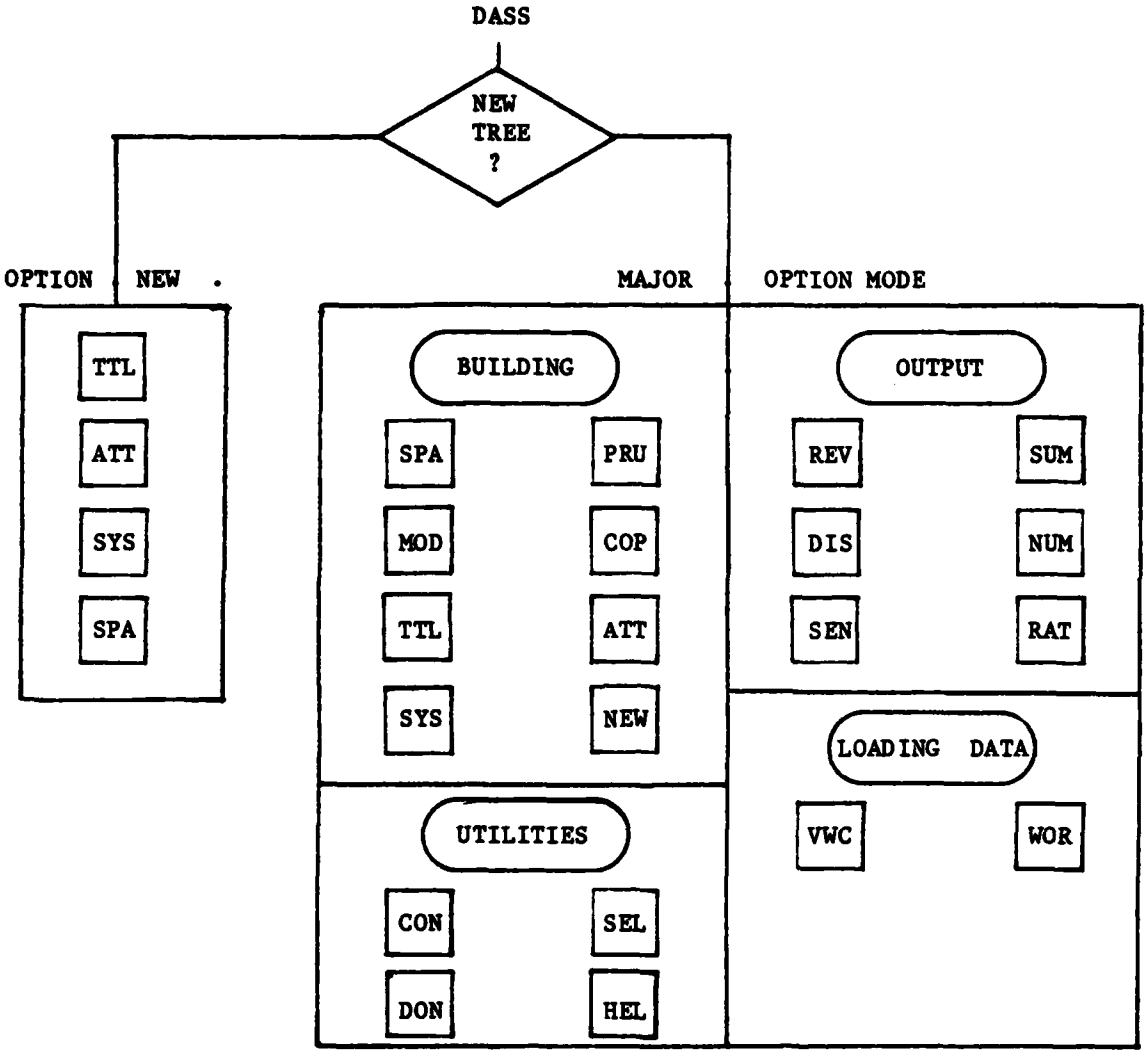


Fig 2. The Logical Structure of the DASS Program

subject, requesting source, and other data which might be useful in identifying the reasons and sources for the structure.

The next mode entered is the ATTribute entry mode, which allows the user to enter a label (up to 10 characters) which describes the nature of the values which will later be entered. This entry is later used to prompt the user. The entry for our sample is 'ABILITY'.

The system labels are entered next, through the SYStems mode. As in the case of the attribute, these entries are used as prompts for the user later in the program. The user might keep in mind the fact that although ten characters are used for some prompts, there are also some cases where only the first five characters are used (to compact the data somewhat). For our sample the entries will be 'F-4', 'F-15', and 'F-111'.

At this point the program will progress automatically to the SPAn mode. In this mode the user is allowed to enter nodes rapidly, by merely entering the labels and letting the program generate the Node Reference Numbers (NRN). The user enters the labels in sets or families, descendent from the current parent node. Every newly entered node will be visited in the span mode, to allow addition of families to them. The input of the first five nodes of the sample would go as follows

```
SELECT TOP NODE FOR SPANNING TREE BUILDING.
SPAN AT ALL NODES? 'Y'
DID YOU WANT TO BUILD A NEW TREE? 'Y'
ADDING DOWNLINKS TO NODE ...
 0 MASTER
LABEL? BEST PLANE
LABEL? (null)
ADDING DOWNLINKS TO NODE ...
 1 BEST PLANE
LABEL? 'AERO'
LABEL? 'SURVIVABLE'
LABEL? 'COMBAT'
```

```
LABEL? 'DONE'                (this is an optional exit to next node)
ADDING DOWNLINKS TO NODE ...
 1  1 AERO
LABEL? 'HIGH ALT'
  (and so forth)
```

After exiting from the span mode the program will close and reopen the data files. By doing this DASS ensures that the work done to this point is on disk, so that if the program abends, the user will not lose the work done so far (see suboption Calculate for a reason for an abend).

Having just created a tree structure, it is desirable to review it. The REView mode will generate a condensed print of the tree structure in the data file, or will print a portion of the data structure (user selects which). A review print of the sample problem would appear as

```
OPTION? 'REV'
SELECT TOP NODE FOR REVIEW.
REVIEW ALL NODES? 'Y'


 1 BEST PLANE
 1  1 AERO
 1  1  1 HIGH ALT
 1  1  2 LOW ALT
 1  2 SURVIVABLE
 1  3 COMBAT
 1  3  1 AIR-AIR
 1  3  2 AIR-GRND
 1  3  3 RECCE
                                        (Ref 3 for this format)
```

In this review, the NRN is given, along with the label for each node.

If the user is dissatisfied with one of the labels, the MODify option will allow the quick repair of that node, as the user will be asked to supply an NRN, and the label input will be placed at the node specified by that NRN.

The next step is to input the values and weights for the structure. First, however, it is possible to get a worksheet generated through the WORksheet mode. This worksheet provides blanks for the hand entry of values and weights, and is useful for collecting all such information prior to starting the entry of the data.

Assuming that a set of inputs is available, they must be entered into the model. This is accomplished through the VWC mode, which controls the entry of Values, Weights, and Calculation of the inner values and cumulative weights.

Entering of values is either done on a node by node basis (with the nodes being selected by NRN), or on a sweep through the tree, where each candidate (data) node is offered for data entry. Again using our example, the values for the first node would be entered through the following series of commands

```
OPTION? 'VWC'
INPUT DATA NODE VALUES? 'Y'
YOU ARE ABOUT TO BE ASKED FOR DATA NODE VALUES.
GET VALUES FOR ALL DATA NODES? 'Y'

GIVEN THE FOLLOWING ... 1  1  1 BEST PLANE-AERO      -HIGH ALT  -
ENTER THE MEASURE OF abilty     FOR EACH SYSTEM
IN THE FOLLOWING ORDER ...
F-4  -F-15 -F-111-
?'45 80 30'
ENTER COMMENTS ON THESE VALUES.
?'USER ENTERS ANY COMMENTS ON THE VALUES JUST ENTERED'
?'5-6 LINES OF ENTRY (DEPENDING ON THE NUMBER OF SYSTEMS) ARE AVAILABLE'
?'AND ENTRY IS TERMINATED BY ENTERING A NULL LINE'
?''  (null)
```

The entry of weights can also be accomplished on a node by node basis, or by sweeping through the tree. These weights are input as either: (1) fractions which sum to 1.0, (2) percentages, which sum to 100.0, or (3) ratios, which are measured against each other. The user

may use any of the three which feels comfortable. The program will automatically normalize the input weights, and echo print them as percentages for verification by the user. Using the example problem, the following are the initial commands encountered when entering all weights

```
OPTION? 'VWC'
INPUT DATA NODE VALUES? 'Y' or 'N' (whichever)
WEIGHT NODES? 'Y'
YOU ARE ABOUT TO ENTER WEIGHTS FOR NODES.
WEIGHT ALL BRANCHING NODES? 'Y'


 1
BEST PLANE-
    FACTOR        WT     F-4     F-15    F-111  CUMWT
 1)AERO         (.00)       I       I        I   0.00
 2)SURVIVABLE*  (.00)   50.00  100.00     0.00   0.00
 3)COMBAT       (.00)       I       I        I   0.00
    TOTAL                   I       I        I   0.00


FACTOR       ( 1) ( 2) ( 3)
NEWWEIGHTS? '1 2 4'
NORMALIZED     14   29   57
ARE THESE WEIGHTS OKAY? 'Y'
ENTER COMMENTS ON THESE WEIGHTS.
?'HERE THE USER MAY ENTER COMMENTS ON THE RATIONALES FOR THE WEIGHTS'
?'JUST AS RATIONALES MAY BE ENTERED FOR THE VALUES WHICH ARE INPUT'
?'' (null)
```

Once the user has entered a complete set of weights and values, the tree is ready to be "collapsed", or "folded back". This is the process which calculates the internal values and the cumulative weights for the nodes. This step is necessary before any output can be used, or any sensitivity analyses be conducted. This process is initiated through the option VWC, and can be directly accessed immediately after entering the values and weights if desired. WARNING: This may cause the user to exceed the allowed CP time for an interactive job. If this is the case, and if no modifications have been made to the structure of the tree, the

systems, or the attribute since the last occurance of the message "NOW CLOSING (SAVING) TAPE FILE n", then nothing will have been lost, IF the user then catalogs, stores or extends the file (as required, see "Saving Data").

The user is now ready to get some output from the model. The output formats vary from raw tabular data to formatted tables and plots. The simplest format is the review with numbers generated by the NUM mode. This display is essentially a review style tree listing, with the weight and value information added. The structure and interpretation of the NUM print is discussed with the detailed discussion of the option, and will not be repeated here.

The next format is generated by either the SUM mode (for large blocks of the tree) or the DIS mode (to display one node at a time) (the source of the tabular portion of this format was reference 3). Again, the details of the information contained in the output are discussed with the detailed discussion of the option. This format is useful for getting a feel for the source of the observed differences in the systems being considered, as the tabular values are "normalized" to each other, such that it makes sense to compare them.

One of the most important capabilities of the program is the sensitivity analysis. As all of the hard numbers entered into the model are subjective evaluations of the systems and the relationships of the nodes, it is crucial that sensitivity analyses be performed. In fact, effective sensitivity analysis is one key to effective decision analysis. When a model is too large, it may be possible, through sensitivity analysis, to determine which portions of the model need refinement, which would allow the concentration of effort where it will

have the highest payoff (Ref 23). One important fact to remember about performing the sensitivity analysis on weights which DASS provides, is that while the mathematics of the sensitivity are exact, the interpretation changes if the range of the sensitivity testing is large. This is because of the nature of the hierarchy, which is that the parts are all linked together, and changes in one area may affect other judgments in seemingly unrelated portions of the model.

That is the end of the primary options available to the user for building, loading and displaying the model. There are several other options which can be useful and these merit discussion, albeit briefly. The first of these is the PRUne option, which allows the user to prune selected branches from the tree structure. This can be useful, to a point, if certain areas of the tree have become essentially irrelevant, and the value of continuing those portions is less than the cost of maintaining good values there.

Another sometimes useful option is the COPy option, which allows the user to copy similarly structured portions of the tree, saving having to enter them separately. This is especially handy in symmetrical trees. Nearly symmetrical tree portions could be copied, then pruned down to the proper configuration.

The last option, which is one which was recommended in the literature, is the RATionale returning option, which allows the user to get a printout of the comments which are entered when the weights or values are entered for a node(Ref 14). This is handy when a model is returned to after some absence, or when looking at a model constructed by someone else. Of course, it is up to the user to maintain good habits when entering the comments, as, unlike the numerical inputs, the

lack of comments will not keep the model from working.

Summary

In summary, the user is provided with a quick reference guide, in the form of the figure 4. Additionaly, the user may get a printout of the information in this figure at any time merely by using the HELp option.

III. GETTING UP AND RUNNING

The following sections deal with the interfacing of DASS to the user and the CDC 6600. Although the instructions are fairly complete, they are not intended to replace the CDC documentation, and it is assumed that the user brings some knowledge of the CDC time-sharing procedures to the terminal.

Getting Started

Once the user has determined an initial hierarchical structure, it is time to start using DASS. There are three stages in using DASS: (1) LOGIN, attach any previously created data files, attach the DASS program library, and execute the DASS program, (2) use the DASS program to interactively enter, investigate, and manipulate the model, and (3) terminate DASS execution, save new or modified data files for later use, and LOGOUT.

LOGIN

The user is expected to be able to get logged onto the CDC time-sharing system, and should be at the "COMMAND?" level. The DASS program library must now be attached and declared as a library. This is accomplished using the following

```
PROMPT:  COMMAND?
RESPONSE:  ATTACH,[LIB],DASS,ID=AFIT
PROMPT:  COMMAND?
RESPONSE:  LIBRARY,[LIB]
```

where

[LIB] is a user selected local file name

Getting Files

It is important for the user to be able to leave a problem, and come back to it later without having to reenter the data. DASS is set up to store all the relevant data on disk data sets for later use. DASS will work on either new files (temporary or to become permanent) or previously created files which have been ATTACHed. Up to three separate files, in any combination of old and new, can be used at a time.

Creating a New File. Creating a new, temporary DASS file, to be deleted immediately (at LOGOUT) is accomplished at execution (see "Execution"). Creating a new DASS file, to be saved when finished, is accomplished through the following commands

```
PROMPT:  COMMAND?
RESPONSE:  REQUEST,[DLFN1],*PF
```

where

[DLFN1] is a user selected local file name

Accessing an Old File. Accessing a previously created DASS file is accomplished through the following command

```
PROMPT:  COMMAND?
RESPONSE:  ATTACH,[DLFN2],[DPFN2],ID=[IDPFN2]
```

where

[DLFN2] is the user selected local file name

[DPFN2] is the user selected (at time of cataloging, see "Saving Dass Data") permanent file name

[IDPFN2] is the user selected ID for [DPFN2]

Execution

After the DASS library is attached, and all desired files are created or attached, the user is ready to start the DASS program. This is accomplished by the following command

```
PROMPT:  COMMAND?
RESPONSE:  DASS,[DLFN1],[DLFN2],[DLFN3]
```

where

[DLFNn] is the appropriate user supplied local file name, and is accessed from within the program as TAPE FILE-n

and, if

[DLFNn] is not supplied, then the default name used is TAPEn, which is automatically created if the user selects file n without having supplied [DLFNn]

Saving Data

Upon exiting DASS, it becomes necessary for the user to save (at the system level) the data which was created (if any). If the user does not want to save any new data, or if none was created, then at this point, LOGOUT. If the new data to be saved was created in a previously cataloged file, and was attached for this run, then the following command will save the updated file

```
PROMPT:  COMMAND?
RESPONSE:  EXTEND,[DLFNn]
```

where

[DLFNn] is the local file name used when the user attached the permanent file

Note that if a temporary fix were made, as part of a "what if?" analysis, the user could, by not EXTENDing, prevent the permanent update of the permanent file.

If the new data to be kept was placed in a REQUESTed file (new)

then the following command is necessary to make the file permanent

```
PROMPT:  COMMAND?
RESPONSE:  CATALOG,[DLFNn],[DPFNn],ID=[IDPFNn],RP=[xxx]
```

where

[DLFNn] is the local file name which was requested

[DPFNn] is the desired permanent file name

[IDPFNn] is the desired ID for the new file

[xxx] is the desired retention period for this data set, after xxx days the data set will be lost. xxx=999 will prevent the loss of the data set <u>if</u> it is attached at least every eight (8) days.

IV. THE OPTIONS

The following sections deal with the major options and their suboptions in explicit detail (from the user's standpoint). User's wishing only to be reminded of the functions of the options should look either to the section 'PROGRAM FLOW' or to the quick reference found on the next page (figure 4). The options are discussed in alphabetical order, and each begins on a new page. All are discussed using the same general format, which is shown in figure 3.

```
OPTION:  (option mnemonic) (option name)
USE:  a general discussion of the use of the option
********** C A U T I O N **********
               (cautions)
********** C A U T I O N **********
ROUTINE CALLED:  "_____" (the routine called to execute the
                          option (if any))
PROMPT-n:  (each prompt expected and
RESPONSE-n:  the responses expected are discussed, as a pair)
```

Fig 3. Option Discussion Format

```
YOUR OPTIONS ARE AS FOLLOWS ...

INPUT (ROUTINE USED)  FUNCTION PERFORMED

ATT (RDATT )  ATTRIBUTE LABEL ENTRY (REGRET/VALUE)

CON (CONTRL)  CONTROL FLAG CHANGES (IPRINT,ISAVD,NTAPE)

COP (COPYR )  COPIES ONE NODE TO ANOTHER (INCLUDING
              ALL DOWN BRANCHES)

DIS (DISPLA)  DISPLAY ONE NODE

DON (       ) DONE WITH WORK, CLOSE LAST FILE

HEL (       ) WILL IMMEDIATELY ALLOW REPRINT
              OF THIS MENU

MOD (MODIFY)  MODIFIES EXISTING TREE, NODE BY NODE

NEW (INITT )  NEW TREE BUILDING DRIVER

NUM (NUMREV)  SAME AS REVIEW, BUT WITH WEIGHTS
              AND VALUES THROWN IN.

PRU (PRUNE )  PRUNES THE TREE NODES

RAT (REASON)  RATIONALE PRINTING

REV (REVIEW)  REVIEW PRINT OF TREE

SEL (TAPER )  SELECTS TREE FILE AND OPENS OR
              CLOSES FILES AS NECESSARY.

SEN (SENSTV)  SENSITIVITY ANALYSIS ON WEIGHT OF A BRANCH

SPA (SPAN  )  ADD TO TREE BY ADDING DOWNLINK
              NODES TO CURRENT NODE.

SUM (SUMMRY)  SUMMARY DISPLAYS OF TREE OR NODE AND
              ITS BRANCHES (MULTIPLE DISPLAYS)

SYS (RDSYS )  SYSTEM LABEL ENTRY

TTL (RDTTL )  DATA FILE TITLE ENTRY

WOR (WRKSHT)  WORKSHEET GENERATOR

VWC (WVLOAD)  DRIVER FOR LOADING WEIGHTS AND VALUES,
              AND RECALCULATES TREE VALUES.

      ANY OTHER ENTRY IS IGNORED (UP TO THREE TRIES).
```

Fig 4. The Major Options

OPTION: ATT (attribute)

USE: This option is used to enter a label for the attribute which is being used. Reasonable responses include 'value', 'regret', or 'deficiency'. This response is later used to prompt the user when asking for the values for the systems/options being considered.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "RDATT"

PROMPT-1: ENTER ATTRIBUTE (REGRET OR VALUE)?
RESPONSE-1: up to ten (10) characters of name for the type of attribute being used.

OPTION: CON (controls)

USE: Used to modify certain control variables within the program. The three variables are:

IPRINT - the debug print flag which turns the debug print on and off

NTAPE - the selection variable which points to the FORTRAN unit number for the tree file now being accessed

ISAVD - a variable used to flag the condition of the data now in the program (saved or not saved)

********** C A U T I O N **********

1. CHANGING THE FILE NUMBER WILL POSSIBLY LOSE ALL THE DATA IN THE PROGRAM

2. SETTING THE DEBUG PRINT FLAG TO ON (1) WILL GENERATE LARGE AMOUNTS OF PRINT IF THE OPTION 'MOD' IS USED

********** C A U T I O N **********

ROUTINE CALLED: "CONTRL"

PROMPT-1: SELECT...CONTROL VARIABLE TO MODIFY(0-3)?
RESPONSE-1: integer (0-3) which chooses the variable to be modified.
The options are:
0 - program control reverts to the main option mode
1 - change "IPRINT", the debug print control flag
2 - change "NTAPE", the file selection pointer
3 - change "ISAVD", the flag on the condition of the data in the program
OTHER ENTRIES - other entries (integers out of range 0-3) are met with prompt-3

PROMPT-2: OLD VALUE OF variable name WAS value of variable ,
NEW VALUE IS?
RESPONSE-2: user responds with the desired new value of the variable selected

PROMPT-3: YOUR OPTIONS ARE ...
0)NONE OF THESE
1)IPRINT 2)NTAPE 3)ISAVD
SELECT OPTION(0-3)?
RESPONSE-3: Same as for prompt-1

OPTION: COP (copy)

USE: This option is used to copy the descendent structure of one node onto a second node.

********** C A U T I O N **********

1. THE NODE WHICH IS BEING COPIED TO MUST NOT BE A DESCENDENT OF THE NODE WHICH IS BEING COPIED FROM

********** C A U T I O N **********

ROUTINE CALLED: "COPYR"

PROMPT-1: ENTER NODE TO BE COPIED TO.
ENTER ... NRN?
RESPONSE-1: User enters the Node Reference Number (NRN) of the node to which to copy. This node must already exist, or an error message will result.

PROMPT-2: ENTER NODE TO BE COPIED.
ENTER ... NRN?
RESPONSE-2: User enters the NRN of the node from which to copy the descendent structure. The descendent structure of this node is copied to the node specified in response-1. Entry of a non-existent node will cause program control to revert to the main option mode.

OPTION: DIS (display)

USE: This option will display the information at a node in both tabular and plotted format. The form of the data is as in figure 5.

```
 1
BEST PLANE-
   FACTOR        WT     F-4     F-15    F-111   CUMWT
 1)AERO         (.14)  34.50   31.00   47.50     .14
 2)SURVIVABLE*  (.29)  50.00   80.00   40.00     .29
 3)COMBAT       (.57)  50.00   48.57   34.29     .57
   TOTAL                47.79   55.04   37.81    1.00


BRANCH      0--------20--------40--------60--------80-------100
AERO        +         +      B A +   C     +         +         +
SURVIVABLE* +         +          C    A     +         B         +
COMBAT      +         +        C +    B     +         +         +
BRANCH      0--------20--------40--------60--------80-------100
```

Fig 5. Display Format

The table portion is a matrix, indexed to row by the NRN digit and node label and to column by the system/option input. The column marked 'WT' represents the normalized weight of the node relative to its siblings, and should always sum to 1.0 . The column marked 'CUMWT' represents the cumulative weight of the node relative to the remainder of the tree. The star (*) found after the label 'SURVIVABLE' indicates that the 'SURVIVABLE' branch ends in a data node. The other two nodes are branching nodes, and the values given for them are calculated values. These calculated values are corrected such that if the nodes were converted to data nodes, and the calculated values were entered as user inputs, the remainder (up-level) of the tree would remain unchanged.

The 'TOTAL' row is the weighted sum of the node values, and would also be the displayed value if this displayed node were shown as part of a higher level node. The plot contains the same information, but in a graph format.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "DISPLA"

PROMPT-1: ENTER NODE TO BE DISPLAYED.
ENTER ... NRN?
RESPONSE-1: Enter the NRN for the node to be displayed.

OPTION: DON (done)

USE: This is the last option used. This signals that the user is finished with the program and the data files. The program will save the data into the files as required, and execution will terminate.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "TSAVE"

MESSAGE-0: JUST CLOSED (SAVING) TREE FILE n.

STOP END OF PROGRAM RUN

OPTION: HEL (help)

USE: This option is to aid the user. Execution of this option will result in the writing of a menu which will list all the major options and a brief description of their use (see figure 4).

********** C A U T I O N **********

1. THIS OPTION WILL WRITE 50 LINES OF OUTPUT (ABOUT ONE PAGE)

********** C A U T I O N **********

ROUTINE CALLED: (none)

PROMPT-1: DO YOU NEED TO SEE THE MENU?
RESPONSE-1: Y(YES) or N(NO), Any other response is met with prompt-2.

PROMPT-2: I ONLY UNDERSTAND Y(YES) OR N(NO).
followed by prompt-1

OPTION: MOD *(modify)*

USE: This option is used to modify the tree one node at a time. Each node addressed by NRN is: (1) created, if necessary, along with all its predecessors as necessary, (2) changed in label only if it already exists.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "ADD"

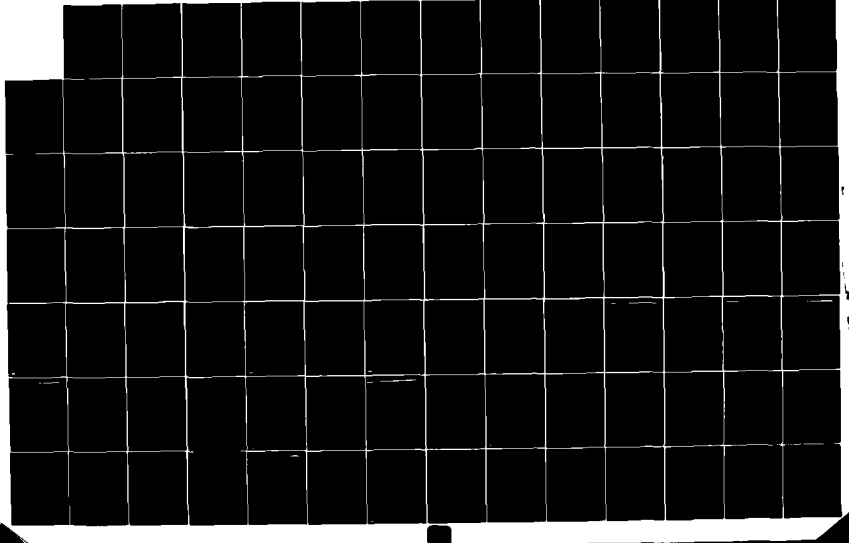
MESSAGE-0: YOU ARE NOW MODIFYING THE TREE. ENTER
A LABEL OF 'DONE' OR '' TO STOP.

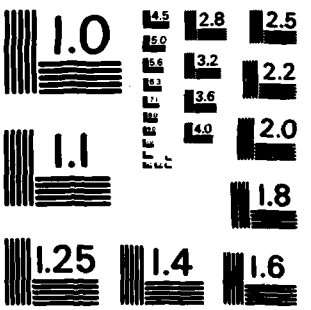
PROMPT-1: LABEL? RESPONSE-1: User may enter a ten (10) character label for the node which is to be specified by NRN. Entering a blank or null string, or 'DONE' will cause the program to exit the option.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User should enter the Node Reference Number (NRN) for the node. The NRN consists of the positive digits normally associated with an NRN, plus a terminal digit of 0. Thus node 1.1.45 would be entered as 1,1,45,0 or as 1 1 45 0 (either is acceptable to the unformatted READ found on the CDC 6600).

Entry of an NRN of zero only (0) will result in the same exit as entry of the blank or null string for the label.

AD-A083 706 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A COMPUTER-BASED DECISION ANALYSIS SUPPORT SYSTEM.(U)
DEC 79 B W MORLAN
UNCLASSIFIED AFIT/GOR/SM/79D-6 NL





OPTION: NEW (new tree structure)

USE: This option is used to generate a new tree structure, either in a newly opened file (automatic call by program) or by overwriting an existing data structure.

********** C A U T I O N **********

1. THIS OPTION WILL OVERWRITE THE CURRENT DATA STRUCTURE IMMEDIATELY (NO ROOM FOR ERROR).

********** C A U T I O N **********

ROUTINE USED: "INITT"

PROMPT-1: (see option TTL)

PROMPT-2: (see option ATT)

PROMPT-3: (see option SYS)

PROMPT-4: (see option SPA)

OPTION: MOD (modify)

USE: This option is used to modify the tree one node at a time. Each node addressed by NRN is: (1) created, if necessary, along with all its predecessors as necessary, (2) changed in label only if it already exists.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "ADD"

MESSAGE-0: YOU ARE NOW MODIFYING THE TREE. ENTER
A LABEL OF 'DONE' OR '' TO STOP.

PROMPT-1: LABEL? RESPONSE-1: User may enter a ten (10) character label for the node which is to be specified by NRN. Entering a blank or null string, or 'DONE' will cause the program to exit the option.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User should enter the Node Reference Number (NRN) for the node. The NRN consists of the positive digits normally associated with an NRN, plus a terminal digit of 0. Thus node 1.1.45 would be entered as 1,1,45,0 or as 1 1 45 0 (either is acceptable to the unformatted READ found on the CDC 6600).

Entry of an NRN of zero only (0) will result in the same exit as entry of the blank or null string for the label.

OPTION: NUM (numeric review)

USE: This option is used to generate a REView style print of the tree structure, but with the added data of the weights, cumulative weights, input values, and corrected values for each node and system/option.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "NUMREV" (NUMeric review)

PROMPT-1: SELECT TOP NODE FOR NUMERIC REVIEW.
REVIEW ALL NODES?
RESPONSE-1: User response is Y(yes) or N(no). If response is Y(yes) then the entire tree structure is reviewed. If response is N(no) then the user will receive prompt-2.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User selects the node which will be the top node reviewed. The NUM format is shown in figure 6 (next page), with an explanation of the data given.

	WEIGHT	CUMWT	VALUES OF THE SYSTEMS (USER INPUT EQUIVALENTS) F-4	-F-15	-F-111 -
1 BEST PLANE	1.00	1.00	47.79	55.04	37.81
1 1 AERO	.14	.14	34.50	31.00	47.50
1 1 1 HIGH ALT	.30	.04	45.00	80.00	30.00
1 1 2 LOW ALT	.70	.10	30.00	10.00	55.00
1 2 SURVIVABLE	.29	.29	50.00	80.00	40.00
1 3 COMBAT	.57	.57	50.00	48.57	34.29
1 3 1 AIR-AIR	.43	.24	50.00	90.00	20.00
1 3 2 AIR-GRND	.43	.24	50.00	20.00	40.00
1 3 3 RECCE	.14	.08	50.00	10.00	60.00

Each line in the table contains the following information (the third line is used as an example).

1) NRN LABEL (1 1 1 HIGH ALT). The Node Reference Number (NRN) is given for each node, along with that node's label.

2) WEIGHT (.30). The weight of this node, relative to its siblings (1 1 2 LOW ALT) is given.

3) CUMWT (.04). The cumulative weight of this node relative to the entire tree (its contribution to the top node calculated values) is given.

4) VALUES OF THE SYSTEMS (45.00 80.00 30.000). The values for the systems are given. These are either the input values (if the node is a data node) or calculated values (if the node is an inner, branching node). These calculated values are those which the user would enter if the branching node were to be replaced with a data node, and the values entered as user inputs.

Fig 6. NUM Print Format

OPTION PRU (prune tree)

USE: Prune is used to eliminate selected branches from the tree. The user is given three options for pruning: (1) prune a node and all its descendents, (2) prune only the descendents of a node, or (3) prune selected children of a node.

********** C A U T I O N **********

1. A FORM OF TEMPORARY PRUNING, WHICH DOES NOT ALTER THE TREE STRUCTURE, IS TO USE WEIGHTS OF 0.0 FOR THOSE BRANCHES WHICH WOULD BE PRUNED.

********** C A U T I O N **********

ROUTINE CALLED: "PRUNE"

PROMPT-1: ENTER NODE TO BE PRUNED.
ENTER ... NRN?
RESPONSE-1: User selects the focal node for the pruning. If the user enters a null node (NRN=0) then the program control will revert to the major option mode. If the selected node does not exist then the program will come back with prompt-2, otherwise the next prompt is prompt-3.

PROMPT-2: NODE NOT FOUND, TRY AGAIN?
RESPONSE-2: Yes or no. If user response is yes then repeat prompt-1, if the user response is no then return program control to the major option mode.

PROMPT-3: SELECT ... TYPE OF PRUNING OPERATION?
RESPONSE-3: User selects the type of pruning operation by number.

0) exit without pruning anything
1) prune the selected node (and defacto, its descendents), then come back with prompt-6 or 7 (depending on the circumstances at the parent node)
2) prune all the descendents of the selected node, then come back with prompt-7 (as the node is now a data node)
3) prune only user selected children of the selected node. The program repeats prompt-4 for each descendent of the selected node.

PROMPT-4: CURRENT DESCENDENT NODE IS nrn-digit and label.
CUT THIS DESCENDENT?
RESPONSE-4: Yes or no. This prompt repeats until the option has been given on each child node of the selected node. The program will then come back with prompt-6 (if the selected node still has descendents) or prompt-7 (if all the children of the selected node were cut from the

tree)
OTHER ENTRIES - other entries (integers out of range 0-3) are met with prompt-5

PROMPT-5: YOUR OPTIONS ARE ...
0)NONE OF THESE
1)NODE+DOWN 2)DOWN ONLY 3)SOME DOWN
SELECT OPTION(0-3)?
RESPONSE-5: Same as for prompt-4

PROMPT-6: (see suboption Weight, page 44, followed by prompt-8)

PROMPT-7: (see suboption Valuate, page 43, followed by prompt-8)

PROMPT-8: (see suboption Calculate, page 45)

OPTION: RAT (RATionale recovery)

USE: This option is used to recover rationales which can be entered when entering weights or values for the nodes in the data structure. The user should provide personal reasons for choosing the input values, perhaps giving the references for those numbers which came from some other source.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "REASON"

MESSAGE-0: REASON ... RETRIEVING RATIONALES STORED WHEN INPUTTING WEIGHTS AND VALUES.

PROMPT-1: GET THE RATIONALES FROM ALL THE NODES?
RESPONSE-1: Yes or no. If yes then all stored rationales are retrieved. If no then the program will come back with prompt-2.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User inputs the NRN of the desired node. The program will print the rationales stored for this node (these will concern the weights if this node branches, otherwise they will be about the values entered). The program will repeat this prompt until the user selects a nonexistent node, or responds with a null node (NRN=0).

OPTION: REV (review)

USE: A review of the tree gives a quick print of the tree structure. The NRN and label are given for each node visited, which will be the user selected node and all its descendents. Each line appears as

NRN LABEL

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "REVIEW"

PROMPT-1: SELECT TOP NODE FOR NUMERIC REVIEW.
REVIEW ALL NODES?
RESPONSE-1: User response is Y(yes) or N(no). If response is Y(yes) then the entire tree structure is reviewed. If response is N(no) then the user will receive prompt-2.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User selects the node which will be the top node reviewed. Selection of a nonexistent node will result in the return of program control to the main option mode.

OPTION: SEL (SELect a data file)

USE: This option is used to select the disk file for use as a storage device for the pseudo-arrays. Each file can be accessed without 'crosstalk' between data structures.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "TAPER"

PROMPT-1: THE CURRENT TREE IS NUMBER n.
JUST CLOSED (SAVING) TREE FILE n.
WITH WHICH TREE WOULD YOU LIKE TO WORK?

RESPONSE-1: User inputs the file which is to be accessed. The choices are 1, 2, or 3 . These correspond to local files (CDC 6600) of TAPE1, TAPE2, and TAPE3. They also correspond to the first three local files specified with the "DASS" card, i.e. if the user executed the program with a command of DASS,TREEA,,TREEC then 1=TREEA , 2=TAPE2 (default), and 3=TREEC.

MESSAGE-1: NOW OPENING TREE FILE NUMBER (user's choice)

NOTE-1: If the newly opened file has no data in it, the program will go to option 'NEW' automatically.

OPTION: SEN (SENsitivity analyses)

USE: Used to conduct weight sensitivity analyses. The selected node's cumulative weight (CUMWT) is allowed to range from the user selected minimum to the user selected maximum. The tree is collapsed with the perturbed cumulative weight (using a mathematical trick), and the values at the top node are printed and plotted.

********** C A U T I O N **********

1. USING MIN AND MAX VALUES FAR FROM THE 'CORRECT' VALUE WILL GIVE MATHEMATICALLY CORRECT NUMBERS, HOWEVER THE VALIDITY OF THESE RESULTS MAY BE SUSPECT DUE TO THE SYNERGISTIC INTERACTIONS OF WEIGHTS WITH WEIGHTS AND WEIGHTS WITH VALUES.

********** C A U T I O N **********

ROUTINE CALLED: "SENSTV"

MESSAGE-0: SENSITIVITY ANALYSES FOLLOW.

PROMPT-1: BRANCH FOR WHICH WEIGHT (CUMWT) IS TO BE PERTURBED.
ENTER ... NRN?
RESPONSE-1: User selects node for sensitivity analysis. Selection of a non-existent node result in an exit from the sensitivity option.

PROMPT-2: CURRENT CUMULATIVE WEIGHT IS cumwt MINIMUM CUMWT (0-1) IS?
RESPONSE-2: User enters the low end of the range of interest. This must be in the interval 0 to 1.0. A minimum weight less than 0.0 will cause an exit from the sensitivity option.

PROMPT-3: MAXIMUM CUMWT (0-1) IS?
RESPONSE-3: User enters the high end of the interval of interest. This must be in the interval minweight to 1.0 or the program will exit from the sensitivity option.

Note - the program will continue with prompts 1 through 3 until the user uses one of the exits.

DISPLAY: The sensitivity display is shown in figure 7 (next page), with a discussion of the format.

```
SENSITIVITY ANALYSES FOLLOW.
BRANCH FOR WHICH WEIGHT (CUMWT) IS TO BE PERTURBED.
ENTER ... NRN?'1 2 0'
 1  2 SURVIVABLE
CURRENT CUMULATIVE WEIGHT IS  .29 MINIMUM CUMWT (0-1) IS?'0.0'
MAXIMUM CUMWT (0-1) IS?'1.0'
   WEIGHT  F-4     F-15    F-111
  0.00    46.90*  45.06   36.93
   .10    47.21   48.55*  37.24
   .20    47.52   52.05*  37.54
   .30    47.83   55.54*  37.85
   .40    48.14   59.03*  38.16
   .50    48.45   62.53*  38.46
   .60    48.76   66.02*  38.77
   .70    49.07   69.52*  39.08
   .80    49.38   73.01*  39.39
   .90    49.69   76.51*  39.69
  1.00    50.00   80.00*  40.00

          0--------20--------40--------60--------80-------100
WT =0.00 +         +        C +  B      +         +         +
WT = .10 +         +         C+   B     +         +         +
WT = .20 +         +         C+   A B   +         +         +
WT = .30 +         +         C+   A   B +         +         +
WT = .40 +         +         C+   A     B         +         +
WT = .50 +         +         C+   A     +B        +         +
WT = .60 +         +         C+   A     +  B      +         +
WT = .70 +         +          C    A    +    B    +         +
WT = .80 +         +          C    A    +      B  +         +
WT = .90 +         +          C    A    +       B +         +
WT =1.00 +         +          C    A    +         B         +
          0--------20--------40--------60--------80-------100
```

This display contains the following information.

1) Tabulated values for each system, at the top node, for the two extrema and for 9 equally spaced points internal to the interval

2) The asterisk (*) flags the best (highest value) system for the given cumulative weight.

3) The graphic format is to present the information in a highly visible form. Note that a steep slope would indicate that a system was highly sensitive to the cumulative weight of the selected node. A steep slope coupled with a small interval would indicate an especially sensitive result.

Fig 7. SENsitivity Format

OPTION: SPA (SPAn)

USE: This option is used to create new tree structures by entering the immediate descendents (children) to nodes. The user enters the NRN for the top node to be processed, and NRN's are automatically generated for all additional nodes as the user enters labels for the new nodes. The user is given the option of creating descendents for each new node created.

********** C A U T I O N **********

1. ALL OLD DESCENDENTS AT A NODE ARE LOST IF EVEN ONE NEW DESCENDENT IS ADDED. NOTE THAT IT IS POSSIBLE TO EXIT FROM A NEWLY PROMPTED NODE BY ENTERING A NULL (' '), A RESPONSE OF 'DONE' OR OF 'EXIT'.

2. A YES RESPONSE TO PROMPT-1A WILL DESTROY ALL PREVIOUS TREE STRUCTURE DATA.

********** C A U T I O N **********

ROUTINE CALLED: "SPAN"

PROMPT-1: SELECT TOP NODE FOR SPANNING TREE BUILDING.
SPAN AT ALL NODES?
RESPONSE-1: Yes or no. A response of yes will cause a depth-first traversal of the tree structure, visiting each existing node, plus all newly created nodes. See caution 1 for safety feature (means of exiting from a node without destroying the downlinks). A no response will result in prompt-2.

PROMPT-1A: DID YOU WANT TO BUILD A NEW TREE?
RESPONSE-1A: If the data structures are empty, or if the user selected node doesn't exist, then the program checks for the user's intentions. A yes response will start the span option with a new master (top) node. A no response will result in an exit from the span mode.

PROMPT-2: ADDING DOWNLINKS TO NODE ...
nrn and label of the current node
LABEL?
RESPONSE-2: User has three (3) options for response:
1. Null ('') or 'DONE' will conclude the span at this node without overwriting the current downlink structure, and the program will continue at the next node in the traversal, with prompt-2.
2. 'EXIT' will cause the program to exit from the span mode.

3 Entry of a label (up to 10 characters) will result in replacing the previous descendent structure with the new node. The new node will have NRN(new)=NRN(old),1 and the program will continue with prompt-3.

PROMPT-3: LABEL?
RESPONSE-3: The responses are the same as for prompt-2, except that it is no longer possible to save the old descendent structure, and the new nodes are numbered consecutively (NRN,2;NRN,3; etc.).

OPTION: SUM (summary)

USE: This option generates a summary of either (1) the entire tree or (2) a selected node and its descendent structure. The format used is that of the display option (see option DIS).

********** C A U T I O N **********

1. THIS CAN GENERATE A LARGE AMOUNT OF PRINT (ON THE ORDER OF 3*n WHERE n IS THE NUMBER OF NODES BELOW THE SELECTED TOP NODE).

********** C A U T I O N **********

ROUTINE CALLED: "SUMMRY"

PROMPT-1: SUMMARY FOLLOWS.
SUMMARIZE ALL THE NODES?
RESPONSE-1: Yes or no. If the response is no then the program will come back with prompt-2, otherwise it will summarize the entire tree data structure.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User selects a node. The selected node and its descendents only are summarized. To get a summary print of a single node, use the option DIS.

OPTION: SYS (system label entry)

USE: This is used to enter labels for the systems or options being considered. These are later used to prompt the user for values, and as part of the display formats (the first five (5) characters are used in displays).

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "RDSYS"

PROMPT-1: ENTER ... SYSTEM n LABEL?
RESPONSE-1: User responds with the n-th system label (up to ten (10) characters, remembering that the first five characters are all that is used in the display formats. This prompt repeats until user enters a null ('') or a label of 'DONE'.

MESSAGE-1: SORRY ... YOU MUST ENTER AT LEAST ONE SYSTEM.
This is followed by prompt-1.

OPTION: TTL (title)

USE: This option elicits a title for later use as a header to the summary output.

********** C A U T I O N **********

(there are no cautions associated with this option)

********** C A U T I O N **********

ROUTINE CALLED: "RDTTL"

MESSAGE-1: ENTER A TITLE FOR THIS DATA STRUCTURE.

PROMPT-1: ?
RESPONSE-1: Enter up to 80 characters of title. This prompt repeats until the user enters a blank line, or until all available title space is used up (usually 5 or 6 lines).

OPTION: VWC (Valuations, Weights, and Calculation)

USE: This option is used to access suboptions for entering weights, entering valuations, or (re)calculating interior values. These options are strongly related, which is why they are grouped under a single major option.

********** C A U T I O N **********

1. RECALCULATING A LARGE TREE MAY TAKE A LONG TIME. THIS TIME MAY BE IN EXCESS OF THE CP (CENTRAL PROCESSOR) TIME REMAINING. IT IS OFTEN WISE TO SELECT THE CURRENT FILE (THIS WILL SAVE YOUR DATA IN THE EVENT THAT THE CP LIMIT IS EXCEEDED, CAUSING AN ABEND).

********** C A U T I O N **********

ROUTINE CALLED: "WVLOAD"

PROMPT-1: INPUT DATA NODE VALUES?
RESPONSE-1: Yes or no. See suboption valuate (next page).

PROMPT-2: WEIGHT NODES?
RESPONSE-2: Yes or no. See suboption weight (next page plus one).

PROMPT-3: (RE)CALCULATE TREE?
RESPONSE-3: Yes or no. See suboption calculate (next page plus two).

SUBOPTION: Valuate (selected from option VWC)

USE: This suboption is used to enter the system/option valuations at the data nodes.

********** C A U T I O N **********

1. THE USER MUST UNDERSTAND THE MEANING OF THE VALUES BEFORE ENTERING THEM. THEY CAN BE SCALED TO ANY REASONABLE SCALE (100 IS A GOOD CHOICE), AS LONG AS THEY ARE CONSISTENT.

********** C A U T I O N **********

ROUTINE CALLED: "VLOAD"

MESSAGE-0: YOU ARE ABOUT TO BE ASKED FOR DATA NODE VALUES.

PROMPT-1: GET VALUES FOR ALL DATA NODES?
RESPONSE-1: Yes or no. If response is no then the program will come back with prompt-2, otherwise prompts 3 and 4 will occur for every data node in the tree.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User selects a node. The program responds with prompts 3 and 4, then repeats prompt-2. If the selected node does not exist, then the program will exit from the suboption valuate. If the selected node is not suitable for valuation (if it is a branching node) then repeat prompt-2.

PROMPT-3: GIVEN THE FOLLOWING ... nrn and label for this node
ENTER THE MEASURE OF attribute FOR EACH SYSTEM
IN THE FOLLOWING ORDER ...
system-1 system-2 ... system-n labels
?
RESPONSE-3: User responds with:
(1) appropriate valuations,
(2) end-of-file (%EOF),
(3) some valuations then end-of-file.
If the user enters values for all the systems then the program will respond with prompt-4. If the user responds with an end-of-file then those values input prior to the end-of-file will be updated, and the program will exit from the suboption Valuate.

PROMPT-4: ENTER COMMENTS ON THESE VALUES.
PROMPT-4A: ?
RESPONSE-4: User enters comments (up to 80 characters per line, between 5-6 lines) until prompt-4A fails to appear or until the user enters a blank (null) line.

SUBOPTION: Weight (selected from option VWC)

USE: To input weights at nodes which have descendents.

********** C A U T I O N **********

1. INPUT OF A NEGATIVE WEIGHT IS NOT DETECTED, HOWEVER IT WOULD INVALIDATE ALL THE CALCULATIONS WHICH USE IT.

********** C A U T I O N **********

ROUTINE CALLED: "WLOAD"

PROMPT-1: WEIGHT ALL BRANCHING NODES?
RESPONSE-1: Yes or no. If response is no the program will come back with prompt-2. If response is yes, then prompts-3, 4, and 5 will occur for every branching node in the tree.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User selects a node. The program responds with prompts 3, 4, and 5, then comes back to prompt-2 after user has entered weights. If the node selected is not suitable for weighting (not a branching node) then repeat prompt-2. If the node does not exist, or if the user inputs a zero (0) then control reverts to the option VWC .

PROMPT-3: (DISplay of the node, without plot)
FACTOR (1)(2)...(n)
PROMPT-3A: NEWWEIGHTS?
RESPONSE-3: User enters the desired weights in any of three forms (all numbers $\geq$ 0.0). The forms are:

1) as percentages (SUM(weights)=100)
2) as fractions (SUM(weights)=1.0)
3) as relative ratios (SUM(weights)=unknown)
4) %EOF (an end-of-file will cause an immediate exit from the weight suboption)

The sum of the entered weights can be 0 (if all weights entered for the current node are 0). The numbers input are separated by commas or blanks. The program will not continue until enough weights have been entered (or until an end-of-file has been entered). If all the needed weights are entered, then the program will come back with prompt-4.

PROMPT-4: NORMALIZED w1 w2 ... wn (weights are internally normalized to sum to 1.0, but are displayed as percentages (to sum to 100))
PROMPT-4A: ARE THESE WEIGHTS OKAY?
RESPONSE-4: Yes or no. If the weights are correct, then continue with prompt-5, otherwise go back to prompt-3A.

PROMPT-5: ENTER COMMENTS ON THESE WEIGHTS.
PROMPT-5A: ?
RESPONSE-5: User enters comments (up to 80 characters per line, between

SUBOPTION: Calculate (selected from option VWC)

USE: Used to calculate the valuations for the internal (branching) nodes. This is often called 'collapsing' the tree.

********** C A U T I O N **********

1. USER IS ADVISED TO SELECT THE CURRENT DATA STRUCTURE BEFORE EXECUTING THIS SUBOPTION AS THIS OPTION MAY ABEND IF THE CP (CENTRAL PROCESSOR) TIME LIMIT IS EXCEEDED.

********** C A U T I O N **********

ROUTINE CALLED: "CALC"

MESSAGE-1: INTERIOR TREE VALUES BEING CALCULATED.

OPTION: WOR (worksheet generator)

USE: This is used to generate a worksheet for use in collecting weights and values for later entry into the program. An example of a worksheet is given on the next page. The user may opt for a worksheet for the entire tree, or for one node plus its descendents.

********** C A U T I O N **********

1. THIS CAN GENERATE LARGE AMOUNTS OF PRINT

********** C A U T I O N **********

ROUTINE CALLED: "WRKSHT"

PROMPT-1: WORKSHEET GENERATION FOLLOWS.
DO YOU WANT A WORKSHEET FOR THE ENTIRE TREE?
RESPONSE-1: Yes or no. If the response is no the program will come back with prompt-2.

PROMPT-2: ENTER ... NRN?
RESPONSE-2: User selects the top node of the portion of the tree for which a worksheet is desired. Selecting a nonexistent node will cause the program control to revert to the major option mode.

```
OPTION?:  WOR
WORKSHEET GENERATION FOLLOWS.
DO YOU WANT A WORKSHEET FOR THE ENTIRE TREE? 'Y'
 1 BEST PLANE
    1 AERO          (WT:____)
    2 SURVIVABLE    (WT:____)
    3 COMBAT        (WT:____)


 1  1 AERO
       1 HIGH ALT     (WT:____)
       2 LOW ALT      (WT:____)


 1  1  1 HIGH ALT

SYSTEMS = -F-4        -F-15       -F-111      -

ABILITY   - (____) - (____) - (____) -


 1  1  2 LOW ALT

SYSTEMS = -F-4        -F-15       -F-111      -

ABILITY   - (____) - (____) - (____) -

                (and so forth)
```

Fig 8. WORksheet Output

APPENDIX C

Programmer's Guide

for

DASS - A Decision Analysis Support System

DASS - DECISION ANALYSIS SUPPORT SYSTEM

PROGRAMMER'S GUIDE

Bruce W. Morlan
Captain USAF

This programmer's guide was written by Captain Bruce W. Morlan as part of the documentation for the Decision Analysis Support System (DASS) program written as part of a Master's thesis for the Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Table of Contents

List of Figures

Introduction

This programmer's guide was written to supplement the internal comments which document the Decision Analysis Support System (DASS). This extra documentation provides: (1) a brief discussion of the algorithm for sensitivity analysis found in the DASS program, (2) a discussion of the hierarchical structure manipulating portions of the program, and (3) a variable and subroutine cross reference map which shows where each variable is used and modified, as well as giving a brief description of the purpose of the variable.

Program Objective

DASS was written to aid in decision analyses of problems which are characterized by hierarchically structured data. This includes Mission Area Analyses (MAA), Management by Objectives (MBO), and Multiattribute Utility problems (MAU). It is based on previous work done for the Defense Advanced Research Projects Agency (DARPA) by Decisions and Designs, Inc. (DDI), McLean, VA.

The Nature of the Problem

To understand the workings of this program, it is helpful to have an understanding of the nature of the problems which it is designed to help solve. Consider the tree structure shown in figure 1, where each node is marked as

LABEL (NRN) + WEIGHT (CUMULATIVE WEIGHT)

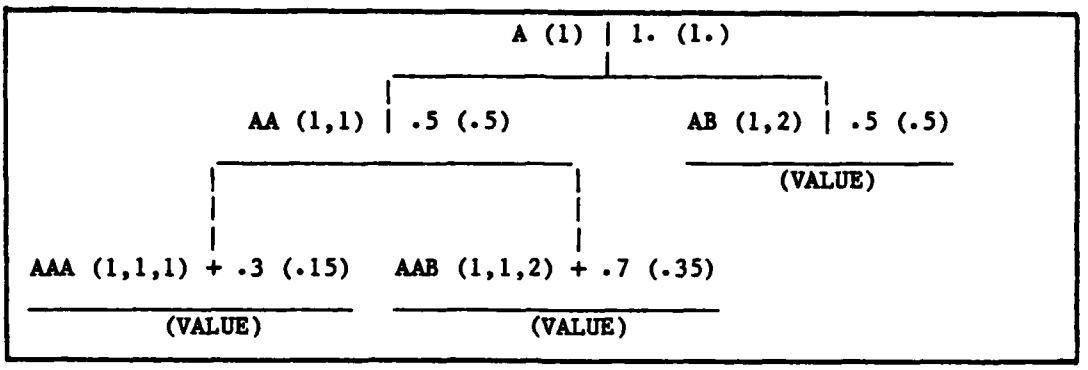


Fig 1. A Hierarchy

In this example, the values are measures of the value of a decision, while the weights measure the importance of the branch, and normalize the scales of the values. DASS can aid an analyst in using decision analysis structures by providing routines to build, modify, weight, valuate, and display these hierarchies.

Sensitivity Analyses

Given that the analyst is interested in such a structure, one of the questions which is relevent concerns the impact of the weights (often subjective) on the composite values. Remember that the definition of the composite value is

$$\sum_I((CUMWT_i)x(V_i))$$

where

I is the set of all descendent data nodes

$CUMWT_i$ is the cumulative weight of the i-th descendent data node

V_i is the value of the system/option at the i-th node

The analyst is often interested in the sensitivity of the composite values to variations in the weights. Sensitivity analysis can be

conducted on the weights in two ways, either by specifying the range of interest for the cumulative weight of a node or by specifying the range of interest for the relative weight of the node. As the algorithms for the two methods are similar, and DASS currently operates using the first, we will discuss only that method.

We are interested in the effect of replacing the j-th weight ($CUMWT_j$) with a new value, say $CUMWT_j^*$. From the assumptions made when using these models, that is, the use of a normalized tree, we can immediately calculate the new composite value through a sequential process (although the steps are combined in the coding into a single step). The routine in DASS which accomplishes the sensitivity analyses is "SENSTV".

Step one is to remove the old effect of the j-th node from the composite value, V. This is done by calculating

$$V = V - (V_j) \times (CUMWT_j)$$

The next step is to re-weight the remaining value, V, to keep the overall tree structure normalized. The calculation is

$$V = V/(1-CUMWT_j)$$

to remove the old weighting factor, then

$$V = V \times (1-CUMWT_j^*)$$

to re-weight the portion of the tree which is complimentary to the j-th node. Having accomplished this, the final step is to add the modified contribution from the j-th node, by

$$V = V + (V_j) \times (CUMWT_j^*)$$

This completes the calculation, resulting in a value which reflects the value which would result from replacing $CUMWT_j$ with $CUMWT_j^*$.

Branching Versus Data Nodes. Although the previously described method for calculating sensitivities is that found in DASS, the way that DASS stores values for internal, branching nodes forces a slight modification. DASS stores the V_j for internal nodes already weighted, and this weighting factor must be factored out before the method may be applied. This is accomplished through the use of the variable WT , which is left at 1.0 if the j-th branch is a data (terminal) branch, where the values stored are the actual input values. When the subject node is an internal node, the values associated with that node are composite values of the descendent terminal nodes and must be re-weighted to become usable. This is accomplished by setting $WT = CUMWT_j$, and factoring it out during the calculations. This section of coding was written in this manner with the idea that it was better to use this method than it would be to use two different formulae for the two possible nodes (internal and data nodes).

Multinode Sensitivity

One interesting idea which has not been incorporated in DASS is that of multinode sensitivity analyses. The basic one node sensitivity could be modified to replace the sensitivity analysis on the weight of the single node 'j' with a sensitivity on a set of nodes 'J'. The weights of the elements of the set J could be allowed to range over intervals of interest (remembering that the tree must remain normalized) and the compliment to J adjusted to calculate multinode sensitivities.

One could envision applying this technique to several nodes

simultaneously, but displaying the information becomes more difficult as the number of nodes increases. In the two node case, however, the display is simple to envision as a matrix which is indexed by the two perturbed weights, and the entries of which are the best system for the given pair of perturbed weights.

Hierarchy Manipulation

A significant portion of DASS is devoted to the problem of manipulating the hierarchical structure. The routines which do this are general purpose routines, and certain of them warrant discussion in some detail. The routines which are discussed here are the "_RAY-_SET" routines, the "NODIN-FIND" set, and the "PRETOT-PRENEX-NEXT" set of routines. The first set is to access the direct access data for the data cells used in the linked list which forms the tree. The second set reads in an input Node Reference Number (NRN) and finds the closest match to that node. The final set conducts the depth-first traversal of the tree. Each of these sets is discussed in greater detail in the sections that follow.

-RAY and -SET

These routines are to access the data cells which hold the pointers, values, weights, and rationales associated with the hierarchical structure. In general (with the exception of the LABEL-LSET pair), these routines are used in pairs of xRAY and xSET, where xRAY(I) recovers the i-th value of a 'pseudo-array'. The term 'pseudo-array' was coined to describe the effect gained from the use of the xRAY and xSET routines. They are used as if they were arrays in the program. This is possible because it is the nature of the FORTRAN

language, in the CDC 6600 implementation, that it cannot distinguish between an array and a function with the same name until the loader determines that the function is not available ('UNDEFINED EXTERNAL REFERENCE' is the associated error message). Thus, to the user, the pseudo-array 'xRAY' is indistinguishable from an actual array if a value is being taken from the array. However, the routine xSET must called as a subroutine to set the value, and this distinguishes the pseudo-array from a normal FORTRAN array.

The five pairs of pseudo-array generating routines are

(1) ARAY and ASET, which contain the weights (relative and cumulative)

(2) CRAY and CSET, which contain the structure title and the rationales

(3) IRAY and ISET, which contain the pointers and NRN digits

(4) LABEL and LSET, which contain the node labels

(5) VRAY and VSET, which contain the system/option values for the node

These are discussed in greater detail in the "VARIABLES" section.

NODIN-FIND

The routines NODIN and FIND are used to elicit a single node (by NRN) from the user, and find that node. If the selected node does not exist the routines will find the node which is closest in the data structure to where the input node would be. Most options (with the notable exception of MOD) will interpret the non-existence of a node as an indication that the user is finished with the option, and will return program control to the program unit which called that option.

NODIN. Routine NODIN merely reads in the user selected NRN,

stopping when an NRN digit less than 0 is entered. If the user entered an NRN (at least one positive NRN digit), then NODIN will call FIND to search for that node.

FIND. FIND searches the tree for a match to the input node. This is accomplished through a modified breadth-first search. Starting with the top level, a crosslink search is conducted for an NRN digit match at each level. Failure to find a match at a level indicates that the input node would be added as a new crosslink at that level. If a match is found, the next level down is entered and the search continues with the next input NRN digit. If there is no downlink path, then the input node would be added as a downlink to the last matched node.

These routines are used extensively by other options within the program, where the user must select a single node.

PRETOT-PRENEX-NEXT

The routines PRETOT, PRENEX, and NEXT are used to drive a depth-first tour of the tree structure. They are invoked by many of the display options and by the CALC routine. PRETOT and PRENEX are preliminary routines, which initialize the variables used by NEXT during the tour. NEXT then conducts the depth-first tour of the hierarchy, moving one node each time called.

PRETOT. Routine PRETOT sets up the variables in the configuration required for a NEXT directed tour of the entire hierarchical structure. It is often called just prior to PRENEX, allowing the user to opt to: (1) tour the entire tree, or (2) tour the downlink structure from a selected node.

PRENEX. Routine PRENEX uses NODIN to elicit from the user the top node of the portion of the tree to be toured for the driving option

(e.g. REV, NUM). NEXT will then tour that node plus all of its descendents.

NEXT. Routine NEXT conducts the tour, one node at a time, starting from the node indicated by PRETOT and PRENEX. After the call to the pre-NEXT routines PRETOT and/or PRENEX, the current node in the programs data structure (the node indicated by IFIND) is the top node of the tour. This is why the first call to NEXT is after the processing of the first node by the calling program. The correct sequence for use of these routines is

```
    CALL PRETOT
    IF(NO("PROCESS ALL NODES?",2))CALL PRENEX
100 CONTINUE
    IF(ICONT.EQ.0)GO TO 200

        (process the current node)

    CALL NEXT
    GO TO 100
200 CONTINUE
        (end of processing)
```

Note that the first check to continue or not (ICONT.EQ.0) is made before the processing of the first node. This allows the user to opt out of the process by selecting a non-existent node (e.g. NRN=0).

The flow for the NEXT routine used in the FORTRAN version was developed directly from the process that a person would use in conducting a tour of a tree manually. Figure 2 shows the flowchart for this algorithm. Figure 3 shows the flowchart for the improved algorithm used in the Applesoft II BASIC program. This improved version is slightly cleaner, in that it uses less storage and is more direct. Implementing the improved algorithm in the FORTRAN version would involve the addition of a variable to mark the level of the node selected in PRENEX and leaving the entire branch to that node in the arrays.

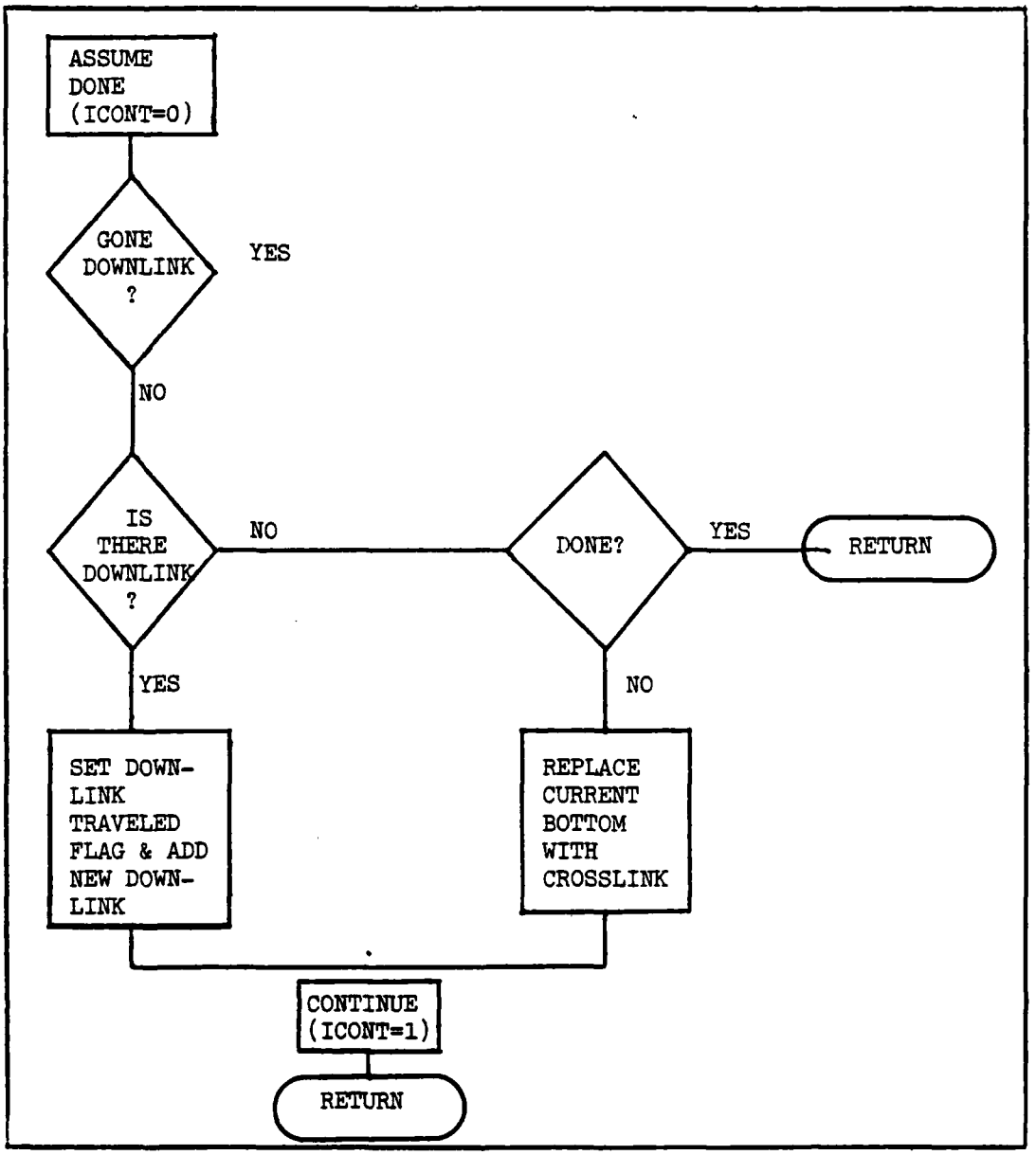


Fig 2. FORTRAN Version of NEXT

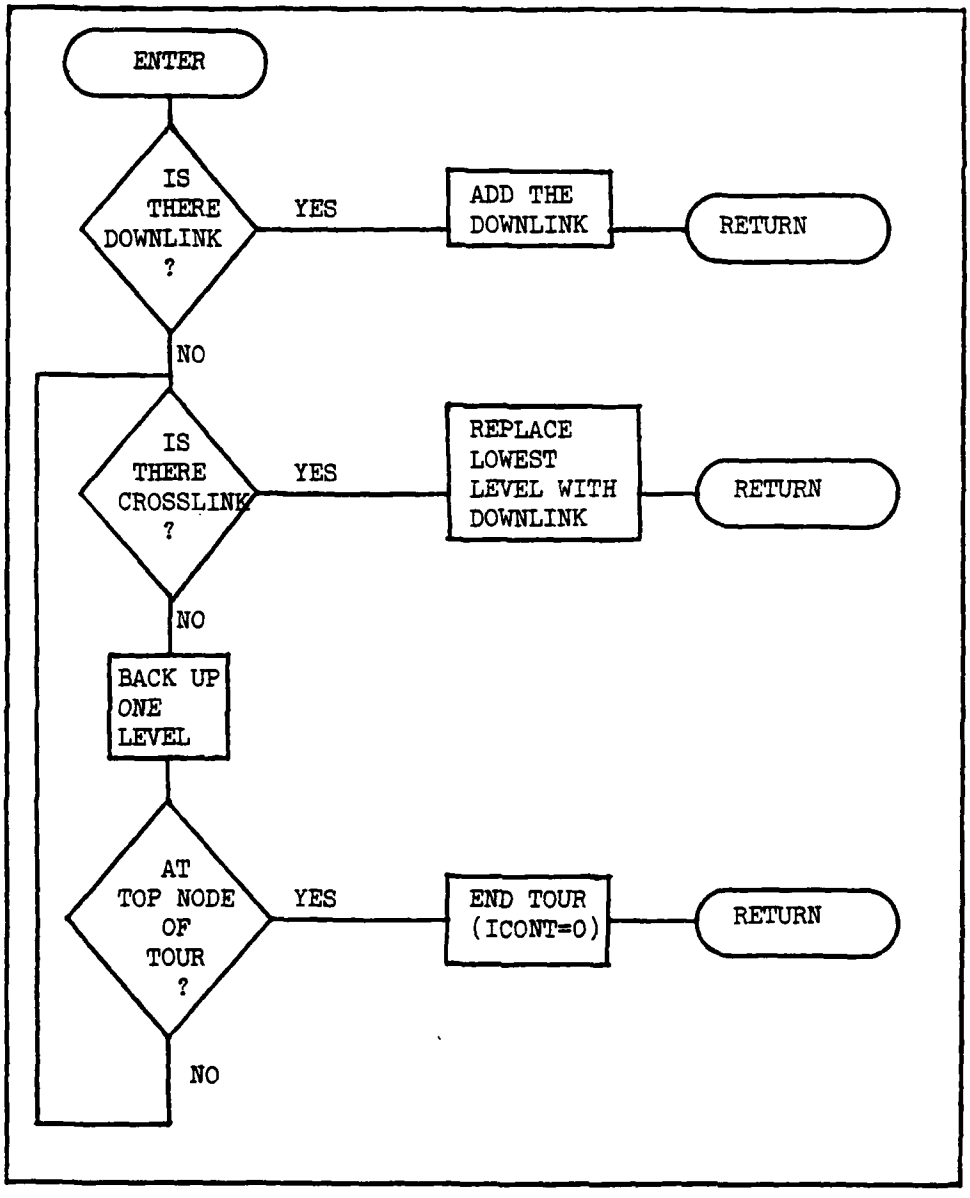


Fig 3. Improved Version of NEXT

Currently the FORTRAN program places only the selected node in the storage arrays.

Variables

This section consists of several parts. It contains a listing of the variables, with a brief discussion of the use of each within the program. Included is a discussion of the use of 'pseudo-variables'. There are three lists which define variables: (1) a list of the global variables (found in labeled COMMON), (2) a section on pseudo-variables, and (3) the local variables (not found in COMMON blocks).
Also supplied is an alphabetical listing of all the variables used, with cross references to the routines which use them. The routines listed under the "USED" label are those which use the variable without changing the value. Those routines listed under the "MODIFIED" label may change the value of the variable.

VARIABLES USED

ARRAY(*=64)	CONTAINS THE DATA FOR CELL(ISIN), IT IS EQUIVALENCED TO PSUEDO-ARRAYS AS FOLLOWS ARRAY(1-4) ARE EQUAL TO IRAY(ISIN,1-4) ARRAY(5-6) ARE EQUAL TO ARAY(1-2) ARRAY(7) IS EQUAL TO LABEL(ISIN) ARRAY(8-(NSYS+8)) ARE FOR VRAY(ISIN,J) ARRAY(THE REST) ARE FOR THE RATIONALES
ATTDUM(*)	DUMMY ARRAY TO FILL OUT THE DIRECT ACCESS RECORD BLOCK CONTAINING THE ATTRIBUTE LABEL (LATT)
ICONT	FLAGS WHETHER TO CONTINUE OR NOT IN TRAVERSING TREE CREATED BY PRETOT-PRENEX-NEXT GROUP OF ROUTINES TO TRAVERSE TREE 0 = DO NOT CONTINUE WITH PROCESSING 1 = CONTINUE PROCESSING
IDATA	FLAGS WHETHER CURRENT NODE(IFIND) IS A DATA NODE 0 = NODE IS NOT A DATA NODE (IT BRANCHES) 1 = NODE IS A DATA NODE (NO BRANCHES)
IFADD	FLAGS RELATIONSHIP OF IFIND NODE TO INPUT NODE, POINTS HOW TO ADD THE INPUT BRANCH TO THE EXISTING BRANCH (AS A DOWNLINK OR A CROSSLINK). 2 = IFIND IS PARENT TO INPUT NODE 3 = IFIND IS BROTHER TO INPUT NODE
IFIND	POINTS TO CELL CONTAINING THE TERMINAL OF THE BRANCH OF NODES WHICH MATCH THE INPUT NRN VECTOR
INNRN(**)	CONTAINS THE INPUT NRN VECTOR (NUMBERS)
IPRINT	DEBUGGING PRINT FLAG 0 = NO DEBUG PRINT (DEFAULT) 1 = YES, GIVE DEBUG PRINT
ISAVD	FLAGS IF MASTER ARRAY HAS BEEN SAVED (ON THE CDC 660 COMPUTER SYSTEM, IF ISAVD=1 THEN A SYSTEM CRASH WILL NOT CAUSE ANY TREE DATA TO BE LOST (MORE OR LESS)) 0 = NOT YET SAVED 1 = YES, SAVED
ISIN	DATA CELL CURRENTLY IN CORE IS CELL(ISIN). (THAT IS, ARRAY(ISIN) IN IN CORE) USED TO PREVENT UNNECESSARY READING OF DISK DATA

ITOTL	FLAGS TYPE OF TREE TRAVERSAL 0 = NOT A TOTAL TREE TRAVERSAL 1 = TOTAL DOWN NODE TRAVERSAL (FROM INPUT NODE ON DOWN)
LATT	CONTAINS THE LABEL FOR THE ATTRIBUTES (REGRET OR VALUE)
LEVEL(**,3)	VECTOR OF NODES WHICH MAKE UP THE BRANCH ENDING IN THE NODE IFIND (*,1) = LOCATION OF CELL CONTAING NODE * (*,2) = FLAGS WHETHER VISITED OR NOT BY ROUTINE SET PRETOT-PRENEX-NEXT (*,3) = USED IN COPYING, CONTAINS THE POINTER TO THE NEW BRANCH MATCH TO THE OLD BRANCH NODE(IFIND)
LSYS(*=63)	CONTAINS THE LABELS FOR THE SYSTEMS.
LVL	LENGTH OF LEVEL NRN VECTOR (FROM FIND)
MASTER(***)	THIS ARRAY IS USED IN THE CDC6600 DIRECT ACCESS ROUTINES (SEE CDC DOCUMENTATION) (***-4) IS THE MAXIMUM NUMBER OF NODES ALLOWED IN THE TREE.
NDEEP	THE DEPTH OF THE TREE STRUCTURE (MAXIMUM NUMBER OF LEVELS, MAXIMUM NLVLS). SET BY CALC FOR USE BY NUMREV, TO HELP ALIGN THE PRINT.
NDIFF	NUMBER OF LEVELS NOT MATCHED IN INNRN VECTOR (LENGTH INPUT VECTOR) - (LENGTH OF IFIND VECTOR)
NLOUD	GIVES THE SIZE OF THE ARRAY 'MASTER'
NLVLS	CONTAINS THE LENGTH OF THE INPUT NRN VECTOR (LEVEL)
NNODES	NUMBER OF NODES IN TREE PLUS ONE (MASTER NODE) ALSO USED AS A FLAG WHEN OPENING A NEW DATA FILE IF THERE IS NO TREE STRUCTURE IN THE DATA FILE, NNODES IS SET TO 0 BY TLOAD. IT IS THEN CHECKED BY ROUTINES DRIVER AND TAPER TO SEE IF IT IS NECESSARY TO GO DIRECTLY TO OPTION 'NEW'.
NSYS	NUMBER OF SYSTEMS TO BE CONSIDERED
NTAPE	INDICATES THE DATA FILE WHICH IS NOW BEING WORKED WITH (FORTRAN UNIT #).
TITLE(*)	DUMMY ARRAY TO FILL OUT THE DIRECT ACCESS RECORD BLOCK CONTAINING THE NUMBER OF NODES (NNODES)

* THESE ARRAYS ARE DIMENSIONED AT 63 (ATTDUM, LSYS, TITLE)
OR 64 (ARRAY) BECAUSE THEY ARE READ DIRECTLY (USING DIRECT ACCESS METHODS), AND 64 IS A LOCALLY EFFICIENT NUMBER OF WORDS TO READ USING THOSE DIRECT ACCESS METHODS ON THE CDC 6600.

** THESE DIMENSIONS WERE ARBITRARILY SET AT 20. THEY REPRESENT THE MAXIMUM NUMBER OF LEVELS THAT A TREE MAY HAVE. IF THE PROBLEM THAT YOU ARE WORKING ON REQUIRES MORE LEVELS, THESE ARRAYS MUST BE REDIMENSIONED ACCORDINGLY.
(NOTE: THERE IS NO INTERNAL CHECK TO INSURE THAT THE USER DOES NOT GO BEYOND THE DEPTH ALLOWED BY THESE DIMENSIONS)

*** THIS DIMENSION SETS THE MAXIMUM NUMBER OF NODES WHICH THIS PROGRAM WILL HANDLE. IT WAS SELECTED TO ALLOW THE PROGRAM TO RUN INTERACTIVELY

PSUEDO-VARIABLES

PSUEDO-VARIABLES ('ARAY','CRAY','IRAY','LABEL','VRAY')
ARE THOSE PIECES OF DATA WHICH ARE BEST UNDERSTOOD
IF TREATED AS ARRAYS, BUT WHICH MUST, DUE TO CORE
LIMITATIONS, BE HANDLED AS DIRECT ACCESS DATA FILES.
I HAVE ADOPTED THE CONVENTION OF IDENTIFYING THESE
VARIABLES BY ENCLOSING THEIR NAMES IN SINGLE QUOTES
(') WHENEVER I REFER TO THEM. IN THE PROGRAM, THEY
ARE ACCESSED BY USING FUNCTIONS WITH THE SAME NAME.
THEY ARE MODIFIED BY CALLING SUBROUTINES '_SET', WHERE
THE DASH IS FILLED WITH THE FIRST LETTER OF THE VARIABLE
NAME (I.E. SET 'ARAY'(I,J)=WT BY USING
    CALL ASET(I,J,WT)).
THIS CONVENTION WAS ADOPTED FOR TWO REASONS:
(1) THIS MADE PROGRAMMING MUCH EASIER, IN THAT IT
WAS POSSIBLE TO ACT AS IF THERE WAS MUCH MORE CORE
THAN WAS ACTUALLY AVAILABLE, AND (2) IT MAKES
IT POSSIBLE TO CONCENTRATE THE SYSTEM PECULIAR DIRECT
ACCESS ROUTINES IN SHORT, EASILY REPLACED ROUTINES,
ALLOWING EASY TRANSLATION TO OTHER SYSTEMS.

THE PSUEDO-VARIABLES AND THEIR MEANINGS ARE AS FOLLOWS

VARIABLE NAME (FUNCTION)	SET BY (ROUTINE)	EQUIVALENCED TO ARRAY()
'ARAY'(I,J)	ASET(I,J,'ARAY'(I,J))	ARRAY(1-4)
'IRAY'(*,1)	CONTAINS THE NRN NUMBER ASSOCIATED WITH THIS NODE. THESE ARE INPUT INDIVIDUALLY WHEN USING THE MOD(IFY) OPTION, OR ASSIGNED SEQUENTIALLY WHEN USING THE SPA(N) OPTION.	
'IRAY'(*,2)	POINTS TO THE DATA CELL WHICH CONTAINS THE FIRST ENTERED DESCENDENT FROM THIS NODE.	
'IRAY'(*,3)	POINTS TO THE CELL CONTAINING THE FIRST ENTERED CROSSLINK NODE TO THIS NODE.	
'IRAY'(*,4)	POINTS BACK TO THE CELL WHICH POINTS TO THIS NODE (WILL EITHER BE A PARENT OR BROTHER NODE).	
'ARAY'(I,J)	ASET(I,J,'ARAY'(I,J))	ARRAY(5-6)
'ARAY'(*,1)	CONTAINS THE INPUT (NORMALIZED) WEIGHT FOR THIS NODE.	
'ARAY'(*,2)	CONTAINS THE CALCULATED, CUMULATIVE WEIGHT FOR THIS NODE.	

'LABEL'(I)	LSET(I,'LABEL'(I))	ARRAY(7)

'LABEL'(*) CONTAINS THE TEN (10) CHARACTER (CDC 6600) LABEL FOR THIS NODE. THIS IS INPUT BY THE USER, AND IS PRINTED AS NECESSARY.

'VRAY'(I,J)	VSET(I,J,'VRAY'(I,J))	ARRAY(8-(NSYS+7))

'VRAY'(*,J) CONTAINS THE VALUE OF THE J-TH SYSTEM AT NODE(*). IF THE NODE IS A DATA NODE, THIS IS THE VALUE INPUT BY THE USER, ELSE IT IS THE COMPOSITE VALUE AT THIS POINT. IN THIS SECOND CASE, DIVIDING 'VRAY'(*,J) BY 'ARAY'(*,2) WILL RESCALE THE VALUE TO THE SCALE OF THE WORST-BEST VALUE FUNCTIONS.

'CRAY'(*)	CSET(I,'CRAY'(I))	ARRAY((8+NSYS)-64)

'CRAY'(*) CONTAINS THE RATIONALES WHICH ARE ASSOCIATED WITH THIS NODE. THESE ARE INPUT BY THE USER WHEN INPUTTING THE WEIGHTS OR VALUES FOR THIS NODE.

OTHER VARIABLES

AIN(20)	WORK VARIABLE, USED TO TEMPORARILY STORE THE INPUT WEIGHTS WHILE CHECKING FOR USER OPTING OUT (EOF) OR UNTIL THEY CAN BE NORMALIZED.
ALFA(20)	DATA VARIABLE, CONTAINS THE ALPHABET, USED IN THE PLOT ROUTINES TO REPRESENT THE VARIOUS SYSTEMS.
BLANK	DATA VARIABLE, CONTAINS A BLANK (" ")
CUMWT	DATA VARIABLE, CONTAINS THE WORD CUMWT ("CUMWT")
DSPL(51)	WORK VARIABLE, USED TO HOLD THE PLOT LINES WHILE FILLING WITH THE SYSTEM POINTS
DUM(20)	WORK VARIABLE, USED TO FLAG THE OPTIMAL SYSTEM IN THE PRINTOUT FOR SENSITIVITY ANALYSES.
I	WORK VARIABLE, USED AS A DO-LOOP INDEX
IFINDT	WORK VARIABLE, USED TO TEMPORARILY WALK DOWN THE TREE STRUCTURE ONE LEVEL AND IN COPYING NODE TO NODE.
IFROM	WORK VARIABLE, USED TO POINT TO PREVIOUS NODE, OR TO REMEMBER WHETHER DESCENDENT WAS A DOWNLINK OR CROSSLINK TO THE PREVIOUS NODE.

IM	WORK VARIABLE, HOLDS THE VALUE PASSED TO ROUTINE ISET FOR SAVING IN 'IRAY'
IPULL	WORK VARIABLE, USED TO INDEX DATA OUT OF AN ARRAY
IPUT	WORK VARIABLE, USED TO INDEX DATA INTO AN ARRAY
ISTART	WORK VARIABLE, USED TO CALCULATE THE START OF A DO-LOOP (USING I)
ISTOP	WORK VARIABLE, USED TO CALCULATE THE END OF A DO-LOOP (USING I)
ITEM	WORK VARIABLE, EQUIVALENCED TO THE CONTROL VARIABLES (COMMON/CNTRL/). ITEM(I) IS CHANGED IN ROUTINE CONTRL TO CHANGE THE CONTROL VARIABLE WHICH IS EQUIVALENCED TO ITEM(I).
IWORK	WORK VARIABLE, USED AS A GENERAL VARIABLE FOR STORING INTERMEDIATE CALCULATIONS.
IWANT	USED TO SELECT AMONG OPTIONS. USUALLY IN A COMPUTED GO TO.
J	WORK VARIABLE, USED AS AN INDEX FOR A DO-LOOP
JMAX	WORK VARIABLE, USED TO POINT TO THE MAXIMUM ELEMENT IN AN ARRAY NORMALLY INDEXED LOCALLY ON J
JSTART	WORK VARIABLE, USED TO CALCULATE THE START OF A DO-LOOP (USING J)
LABELT	WORK VARIABLE, TEMPORARY STORAGE FOR LABELS USUALLY USED AS THEY ARE INPUT
N	WORK VARIABLE, USED TO HOLD THE LENGTH (IN CDC 6600 DATA WORDS (10 CHAR EACH)) OF THE QUESTION PASSED TO THE ROUTINES YES AND NO.
NCROSS	WORK VARIABLE, USED TO COUNT THE NUMBER OF DESCENDENTS AT THE NEXT LEVEL DOWN FROM A NODE
NNRN	WORK VARIABLE, INCREMENTED DURING SPA(N) MODE TO PROVIDE A UNIQUE NRN FOR EACH NEW DOWNLINK BRANCH ADDED TO A NODE.
NERR	WORK VARIABLE, COUNTS THE NUMBER OF SEQUENTIAL BAD ATTEMPTS TO SELECT AN OPTION (IN DRIVER). RESET WHEN (1) USER SELECTS OPTION HEL(P) (2) USER ERRS THRICE
NPR	DATA VARIABLE, CONTAINS THE NUMBER OF PRUNE OPTIONS. (NPR = 3)

NOPT	DATA VARIABLE, GIVES NUMBER OF OPTIONS AVAILABLE IN DRIVER (NOPT=20)
OPT	DATA VARIABLE, CONTAINS THE 20 THREE-CHARACTER SETS WHICH ARE MATCHED AGAINST THE USER INPUT OPTION.
PRMEN	DATA VARIABLE, HOLDS A SHORT (1 TEN-CHAR CDC 6600 WORD) DESCRIBING EACH PRUNE OPTION.
QUEST	WORK VARIABLE, HOLDS THE QUESTION ARRAY PASSED TO THE ROUTINES YES AND NO.
RSCALE	DATA VARIABLE, CONTAINS THE SCALING FACTOR FOR THE PLOT ROUTINES, (RSCALE = 2.0)
STAR	DATA VARIABLE, CONTAINS A CHARACTER STAR ("*")
USERIN	WORK VARIABLE, USED TO HOLD THE USER'S INPUT OPTION SELECTION, MATCHED AGAINST THE ARRAY OPT.
V	WORK VARIABLE, USED TO HOLD THE USER INPUT VALUE AT A NODE FOR A SYSTEM.
VARBLS	DATA VARIABLE, CONTAINS THE NAMES OF THE CONTROL VARIABLES (COMMON/CNTRL/) WHICH CAN BE MODIFIED BY ROUTINE CONTRL.
VERT	DATA VARIABLE, CONTAINS A CROSS CHARACTER ("+")
VHOLD	WORK VARIABLE, USED TO HOLD AN INTERMEDIATE CALCULATED VALUE PRIOR TO PLACING IN 'VRAY'()
WT	WORK VARIABLE, USED TO HOLD CORRECTION FACTOR FOR VALUES (WT=1.0 FOR DATA NODES, WT='ARAY'(*,2) FOR INTERIOR NODES)
WTDLT	WORK VARIABLE, HOLDS THE STEP SIZE USED IN SENSITIVITY ANALYSES. (WTDLT = (WTMAX-WTMIN)/10.0)
WTMAX	WORK VARIABLE, USED TO HOLD THE MAXIMUM WEIGHT TO BE USED AS A CUMULATIVE WEIGHT (CUMWT) FOR A BRANCH DURING SENSITIVITY ANALYSES.
WTMIN	WORK VARIABLE, USED TO HOLD THE MINIMUM WEIGHT TO BE USED AS A CUMULATIVE WEIGHT (CUMWT) FOR A BRANCH DURING SENSITIVITY ANALYSES,
WTNOW	WORK VARIABLE, USED TO STEP FROM WTMIN TO WTMAX DURING SENSTIVITY ANALYSES
YEANAY	WORK VARIABLE, USED TO HOLD USER INPUT WHEN ASKING YES/NO QUESTIONS.

ZNORM	WORK VARIABLE, USED TO HOLD THE NORMALIZING FACTOR DURING WEIGHT ENTRIES.

CROSS REFERENCE MAP

```
VARIABLE   AIN
                USED: SUM    ,
           MODIFIED: DVIDE   , RDWT   ,

VARIABLE   ALFA
                USED: DSPLOT , SNSPLT ,
           MODIFIED: (NONE) ,

VARIABLE   ARAY
                USED: DSPLAY , DSPLOT , NUMREV , PRNT   , SENSTV ,
                      SNSPLT ,
           MODIFIED: ADD    , ASET   , CALC   , COPYR  , RDWT   ,
                      SPAN   ,

VARIABLE   ARRAY
                USED: (NONE) ,
           MODIFIED: ARAY   , ASET   , CRAY   , CSET   , IRAY   ,
                      ISET   , LABEL  , LSET   , TLOAD  , VRAY   ,
                      VSET   ,

VARIABLE   ATTDUM
                USED: TSAVE  ,
           MODIFIED: TLOAD   ,

VARIABLE   BLANK
                USED: ADD    , DSPLAY , DSPLOT , SENSTV , SNSPLT ,
           MODIFIED: (NONE) ,
VARIABLE   CRAY
                USED: REASON , SUMMRY ,
           MODIFIED: CSET   , RDTTL  , RDWT   ,

VARIABLE   CUMWT
                USED: DSPLAY ,
           MODIFIED: (NONE) ,

VARIABLE   DSPL
                USED: (NONE) ,
           MODIFIED: DSPLOT , SNSPLT ,

VARIABLE   DUM
                USED: (NONE) ,
           MODIFIED: DSPLAY , DSPLOT , SENSTV ,
```

```
VARIABLE   I
              USED  ARAY   , ASET   , CRAY   , CSET   , IRAY   ,
                    ISET   , LABEL  , LSET   , VRAY   , VSET   ,
         MODIFIED:  CALC   , DRIVER , DSPLAY , DSPLOT , DVIDE  ,
                    LISTIT , MENU   , NUMREV , PRNT   , RDV    ,
                    RDWT   , SENSTV , SHEET  , SNSPLT , SPAN   ,
                    SUM    , TLOAD  , TSAVE  , YES    ,

VARIABLE   ICONT
              USED: CALC   , COPYR  , NUMREV , PRNT   , REASON ,
                    REVIEW , SUMMRY , VLOAD  , WLOAD  , WRKSHT ,
         MODIFIED:  NEXT   , NODIN  , PRENEX , PRETOT , SPAN   ,

VARIABLE   IDATA
              USED: CALC   , COPYR  , DSPLAY , DSPLOT , NUMREV ,
                    SHEET  , SUMMRY , VLOAD  , WLOAD  ,
         MODIFIED:  FIND   , NEXT   , PRENEX , PRETOT , PRUNE  ,

VARIABLE   IFADD
              USED: COPYR  ,
         MODIFIED:  ADD    , FIND   , NEXT   , SPAN   ,

VARIABLE   IFIND
              USED: CALC   , COPYR  , DSPLAY , DSPLOT , NUMREV ,
                    RDV    , REASON , SENSTV , SHEET  , SNSPLT ,
         MODIFIED:  ADD    , FIND   , NEXT   , PRENEX , PRETOT ,
                    PRUNE  , RDWT   , SPAN   ,

VARIABLE   IFINDT
              USED: (NONE) ,
         MODIFIED:  COPYR  , DSPLAY , DSPLOT , RDWT   , SENSTV ,
                    SHEET  , SNSPLT , SPAN   ,

VARIABLE   IFROM
              USED: (NONE) ,
         MODIFIED:  DSPLOT , PRUNE  ,

VARIABLE   IM
              USED: ISET   ,
         MODIFIED:  (NONE) ,

VARIABLE   INNRN
              USED: ADD    , FIND   , NUMREV ,
         MODIFIED:  NEXT   , NODIN  , PRENEX , SPAN   ,

VARIABLE   IPRINT
              USED: ADD    ,
         MODIFIED:  CONTRL , DRIVER ,
VARIABLE   IPULL
              USED: (NONE) ,
         MODIFIED:  VRAY   ,
```

```
VARIABLE   IPUT
               USED: (NONE) ,
           MODIFIED: DSPLOT , SNSPLT , VSET   ,

VARIABLE   IRAY
               USED: DSPLAY , DSPLOT , FIND   , LISTIT , NEXT   ,
                     PRENEX , PRETOT , SENSTV , SHEET  , SNSPLT ,
           MODIFIED: ADD    , COPYR  , INITT  , ISET   , PRUNE  ,
                     SPAN   ,

VARIABLE   ISAVD
               USED: (NONE) ,
           MODIFIED: ADD    , CONTRL , PRUNE  , RDATT  , RDSYS  ,
                     TSAVE  ,

VARIABLE   ISET
               USED: (NONE) ,
           MODIFIED: (NONE) ,

VARIABLE   ISIN
               USED: (NONE) ,
           MODIFIED: ARAY   , ASET   , CRAY   , CSET   , IRAY   ,
                     ISET   , LABEL  , LSET   , TLOAD  , VRAY   ,
                     VSET   ,

VARIABLE   ISTART
               USED: (NONE) ,
           MODIFIED: CRAY   , CSET   ,

VARIABLE   ISTOP
               USED: (NONE) ,
           MODIFIED: CALC   , CRAY   , CSET   , TLOAD  ,

VARIABLE   ITEM
               USED: (NONE) ,
           MODIFIED: CONTRL ,

VARIABLE   ITOTL
               USED: REASON , VLOAD  , WLOAD  ,
           MODIFIED: PRENEX , PRETOT , RDV    , RDWT   ,

VARIABLE   IWANT
               USED: PRUNE  ,
           MODIFIED: DRIVER , MENU   ,

VARIABLE   IWORK
               USED: (NONE) ,
           MODIFIED: SENSTV ,
```

```
VARIABLE   J
                USED: ARAY   , ASET   , IRAY   , ISET   , VRAY   ,
                      VSET   ,
             MODIFIED: ADD    , CALC   , CRAY   , CSET   , DSPLAY ,
                      DSPLOT , FIND   , NUMREV , PRNT   , RDV    ,
                      SENSTV , SHEET  , SNSPLT ,

VARIABLE   JMAX
                USED: (NONE) ,
             MODIFIED: SENSTV ,

VARIABLE   JSTART
                USED: (NONE) ,
             MODIFIED: ADD    ,

VARIABLE   LABEL
                USED: DSPLAY , DSPLOT , LISTIT , NUMREV , PRNT   ,
                      PRUNE  , SHEET  ,
             MODIFIED: ADD    , COPYR  , INITT  , LSET   , SPAN   ,
VARIABLE   LABELT
                USED: LSET   ,
             MODIFIED: ADD    , SPAN   ,

VARIABLE   LATT
                USED: RDV    , SHEET  , TSAVE  ,
             MODIFIED: RDATT  , TLOAD  ,

VARIABLE   LEVEL
                USED: CALC   , DSPLAY , LISTIT , NUMREV , PRUNE  ,
                      RDV    , RDWT   , SHEET  , SNSPLT ,
             MODIFIED: COPYR  , FIND   , NEXT   , PRENEX , PRETOT ,
                      SPAN   ,

VARIABLE   LSET
                USED: (NONE) ,
             MODIFIED: (NONE) ,

VARIABLE   LSYS
                USED: DSPLAY , RDV    , SENSTV , SHEET  , TSAVE  ,
             MODIFIED: RDSYS  , TLOAD  ,

VARIABLE   LVL
                USED: CALC   , COPYR  , DSPLAY , LISTIT , NUMREV ,
                      RDV    , RDWT   , SHEET  ,
             MODIFIED: FIND   , NEXT   , PRENEX , PRETOT , PRUNE  ,
                      SPAN   ,

VARIABLE   MASTER
                USED: (NONE) ,
             MODIFIED: TLOAD  ,

VARIABLE   N
                USED: NO     , YES    ,
             MODIFIED: (NONE) ,
```

```
VARIABLE   NCROSS
               USED: DVIDE  ,
           MODIFIED: RDWT   ,

VARIABLE   NDEEP
               USED: NUMREV , TSAVE  ,
           MODIFIED: CALC   , TLOAD  ,

VARIABLE   NDIFF
               USED: ADD    , COPYR  , DISPLA , PRENEX , PRUNE  ,
           MODIFIED: FIND   , PRETOT ,

VARIABLE   NNRN
               USED: SUM    ,
           MODIFIED: SPAN   ,

VARIABLE   NERR
               USED: (NONE) ,
           MODIFIED: DRIVER ,

VARIABLE   NLOUD
               USED: TSAVE  ,
           MODIFIED: TLOAD  ,

VARIABLE   NLVLS
               USED: ADD    , COPYR  , DISPLA , FIND   , NEXT   ,
                     PRENEX , PRUNE  ,
           MODIFIED: NODIN  , PRETOT , SPAN   ,

VARIABLE   NNODES
               USED: CALC   , DRIVER , PRENEX , PRETOT , PRNT   ,
                     SENSTV , TAPER  , TSAVE  ,
           MODIFIED: ADD    , COPYR  , INITT  , SPAN   , TLOAD  ,

VARIABLE   NPR
               USED: MENU   , PRUNE  ,
           MODIFIED: (NONE) ,

VARIABLE   NOPT
               USED: DRIVER ,
           MODIFIED: (NONE) ,

VARIABLE   NSYS
               USED: CALC   , CRAY   , CSET   , DSPLAY , DSPLOT ,
                     NUMREV , RDV    , SENSTV , SHEET  , SNSPLT ,
                     TSAVE  ,
           MODIFIED: RDSYS  , TLOAD  ,

VARIABLE   NTAPE
               USED: ARAY   , ASET   , CRAY   , CSET   , IRAY   ,
                     ISET   , LABEL  , LSET   , TLOAD  , TSAVE  ,
                     VRAY   , VSET   ,
           MODIFIED: CONTRL , DRIVER , TAPER  ,
```

```
VARIABLE   OPT
                USED: DRIVER ,
            MODIFIED: (NONE) ,

VARIABLE   PRMEN
                USED: MENU   , PRUNE  ,
            MODIFIED: (NONE) ,

VARIABLE   QUEST
                USED: NO     , YES    ,
            MODIFIED: (NONE) ,

VARIABLE   RSCALE
                USED: DSPLOT , SNSPLT ,
            MODIFIED: (NONE) ,

VARIABLE   STAR
                USED: DSPLAY , DSPLOT , SENSTV , SNSPLT ,
            MODIFIED: (NONE) ,

VARIABLE   TITLE
                USED: TSAVE  ,
            MODIFIED: TLOAD  ,

VARIABLE   USERIN
                USED: (NONE) ,
            MODIFIED: DRIVER ,

VARIABLE   V
                USED: ASET   , RDV    , VSET   ,
            MODIFIED: (NONE) ,

VARIABLE   VARBLS
                USED: CONTRL , MENU   ,
            MODIFIED: (NONE) ,

VARIABLE   VERT
                USED: DSPLOT , SNSPLT ,
            MODIFIED: (NONE) ,

VARIABLE   VHOLD
                USED: (NONE) ,
            MODIFIED: CALC   , SENSTV , SNSPLT ,

VARIABLE   VRAY
                USED: DSPLAY , DSPLOT , NUMREV , SNSPLT ,
            MODIFIED: CALC   , SENSTV , VSET   ,

VARIABLE   VSET
                USED: (NONE) ,
            MODIFIED: (NONE) ,
```

```
VARIABLE   WT
                USED: (NONE) ,
             MODIFIED: DSPLAY , DSPLOT , NUMREV , SENSTV , SNSPLT ,

VARIABLE   WTDLT
                USED: SNSPLT ,
             MODIFIED: SENSTV ,

VARIABLE   WTMAX
                USED: SNSPLT ,
             MODIFIED: SENSTV ,

VARIABLE   WTMIN
                USED: SNSPLT ,
             MODIFIED: SENSTV ,

VARIABLE   WTNOW
                USED: (NONE) ,
             MODIFIED: SENSTV , SNSPLT ,

VARIABLE   YEANAY
                USED: (NONE) ,
             MODIFIED: YES    ,

VARIABLE   ZNORM
                USED: DVIDE  ,
             MODIFIED: RDWT   ,
```

APPENDIX D

Program Listing for

DASS - A Decision Analysis Support System

THE ROUTINES

```
BASS   (ONLY CALLS DRIVER)
ADD    (ROUTINE TO ADD NODES TO TREE)
ARAY   (ACTS AS AN ARRAY ... W = ARAY(I,J))
ASET   (SETS PSUEDO-ARRAY ARAY(I,J) = V )
CALC   (CALCULATES TREE VALUES FOR NON-DATA NODES)
CONTRL (MENU CONTROLLED MODIFICATION OF CONTROL VARIABLES)
COPYR  (COPIES NODE TO NODE)
CRAY   (ACTS AS AN ARRAY ... "COMMENTS" = CRAY(I))
CSET   (SETS PSUEDO-ARRAY CRAY(I) = "COMMENTS")
DISPLA (DRIVER FOR SINGLE NODE DISPLAYS AND PLOTS)
DRIVER (MAIN DRIVER, USES SELECTION BY NAME, NOT NUMBER)
DSPLAY (GENERATES A SPECIAL FORMAT NODE DISPLAY)
DSPLOT (GENERATES A SPECIAL FORMAT NODE PLOT)
DVIDE  (DIVIDES ELEMENTS OF A VECTOR BY A CONSTANT)
FIND   (ROUTINE TO FIND MATCH TO NODE IN INNRN)
INITT  (NEW TREE BUILDER, CALLS RDATT,RDSYS,RDTTL,SPAN)
IRAY   (ACTS AS AN ARRAY ... IN = IRAY(I,J))
ISET   (SETS PSUEDO-ARRAY IRAY(I,J) = IN )
LABEL  (ACTS AS AN ARRAY ... "LABEL" = LABEL(I,)
LSET   (SETS PSUEDO-ARRAY LABEL(I) = "LABEL")
LISTIT (PRINTS A SPECIAL FORMAT LINE)
MENU   (DRIVER TO OPERATE NUMERIC MENU SELECTION)
NEXT   (TRAVERSES TREE (USED AFTER PRENEX))
NO     (FUNCTION RETURNS .TRUE. (NO) OR .FALSE. (YES))
NODIN  (GETS INPUT NRN, CALLS FIND)
NUMREV (REVIEW STYLE PRINT OF TREE WITH WTS AND VALUES)
PRENEX (SETS UP TO TRAVERSE TREE (ALL OR PART) BY 'NEXT')
PRETOT (PREPARES TO TRAVERSE ENTIRE TREE)
PRNT   (PART OF DEBUGGING PACKAGE)
PRUNE  (CLIPS OFF TREE BRANCHES)
RDATT  (READS IN ATTRIBUTE CHARACTERISTIC (REGRET/VALUE))
RDSYS  (READS IN LABELS FOR THE SYSTEMS)
RDTTL  (READS THE TITLE FOR THE DATA STRUCTURE)
RDV    (READS IN VALUES FOR ONE NODE)
RDWT   (READS WEIGHTS FOR DESCENDENT NODES)
REASON (RETURNS THE RATIONALE FOR WEIGHTS OR VALUES)
REVIEW (PRINTS A REVIEW OF THE TREE)
SENSTV (WEIGHT SENSITIVITY ANALYSIS)
SHEET  (SINGLE LINE OF WRKSHT OUTPUT)
SNSPLT (WEIGHT SENSITIVITY ANALYSIS PLOT)
SPAN   (ADDS NODES BY SPANNING SETS)
SUM    (SUMS ELEMENTS OF A VECTOR)
SUMRY  (DRIVER FOR MASS DSPLAY)
TAPER  (ROUTINE DRIVES TREE FILES)
TLOAD  (TO OPEN MASS STORAGE)
TSAVE  (TO CLOSE MASS STORAGE)
VLOAD  (SUBDRIVER TO GET VALUES FOR DATA NODES)
VRAY   (ACTS AS AN ARRAY ... V = VRAY(I,J))
VSET   (SETS PSUEDO-ARRAY VRAY(I,J)=V )
WLOAD  (DRIVER TO LOAD WEIGHTS FOR NODES)
WRKSHT (GENERATES WORKSHEET)
```

```
WVLOAD (DRIVES WLOAD, VLOAD AND CALC)
YES    (FUNCTION RETURNS .TRUE. (YES) OR .FALSE. (NO))
```

```
CPAGE START DASS   (ONLY CALLS DRIVER)
      PROGRAM DASS(TAPE1,TAPE2,TAPE3,INPUT,OUTPUT)
C  ********************************************************************
C  *                                                                  *
C  *  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  *
C  *  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  *
C  *  XXXXXXXXXXX       XXXXXX    XXXXXX      XXXXX      XXXXXXXXXXXX  *
C  *  XXXXXXXXXXX  XXX  XXXX  XX  XXXX  XXX  XXX  XXX  XXXXXXXXXXXX  *
C  *  XXXXXXXXXXX  XXX  XXX  XXXX  XXX  XXXXXXXX  XXXXXXXXXXXXXXXX  *
C  *  XXXXXXXXXXX  XXX  XXX  XXXX  XXXXX  XXXXXXXX  XXXXXXXXXXXXXX  *
C  *  XXXXXXXXXXX  XXX  XXX        XXXXXXX  XXXXXXXX  XXXXXXXXXXXX  *
C  *  XXXXXXXXXXX  XXX  XXX  XXXX  XXX  XXX  XXX  XXX  XXXXXXXXXXX  *
C  *  XXXXXXXXXXX       XXXX  XXXX  XXXX      XXXXX      XXXXXXXXXX  *
C  *  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  *
C  *  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  *
C  *                                                                  *
C  ********************************************************************
C  ********************************************************************
C  *                                                                  *
C  *  PROGRAM:  DASS                                                  *
C  *  USE:  TO CALL THE MAIN DRIVER ROUTINE                           *
C  *        AND ESTABLISH THE INPUT AND OUTPUT FILES                  *
C  *        AS TAPE1, TAPE2, TAPE3, INPUT(TAPE5), AND                 *
C  *        OUTPUT (TAPE6).  THESE CORRESPOND TO UNITS 1,2,3,5, AND 6*
C  *        FOR FORTRAN READ AND WRITE STATEMENTS.                    *
C  *  CALLED BY: (NONE)                                               *
C  *  ROUTINES CALLED: DRIVER                                         *
C  *  VARIABLES:                                                      *
C  *       USED: (NONE)                                               *
C  *       MODIFIED: (NONE)                                           *
C  *                                                                  *
C  ********************************************************************
C
C     MAIN IS THE ONLY ROUTINE WHICH IS NOT FOUND IN THE
C     COMPILE LISTINGS OR PROGRAM LISTINGS ALPHABETICALLY, EXCEPT
C     THAT THE DIRECT ACCESS ROUTINES ARE FOUND IN PAIRS, WHERE THE
C     RETRIEVAL ROUTINE IS FOUND FIRST, AND THE PAIR ARE FOUND
C     BY THE RETRIEVAL ROUTINE NAME (I.E. IRAY AND ISET ARE FOUND
C     AS IRAY)
C
      CALL DRIVER
      STOP
      END
```

```
CPAGE START ADD    (ROUTINE TO ADD NODES TO TREE)
      SUBROUTINE ADD
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
      DATA BLANK/" "/
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE ADD                                                     *
C  *  USE: TO ADD TO TREE OR MODIFY LABELS OF NODES                      *
C  *       ALREADY IN THE TREE.  NODES ARE ACCESSED                      *
C  *       BY INPUTTING A 'NRN VECTOR', WHICH POINTS THE                 *
C  *       WAY THROUGH THE TREE, LEVEL BY LEVEL.                         *
C  *  CALLED BY: DRIVER                                                  *
C  *  ROUTINES CALLED: NODIN, PRNT                                       *
C  *  VARIABLES:                                                         *
C  *       USED: BLANK , INNRN , IPRINT , NDIFF , NLVLS                  *
C  *       MODIFIED: 'ARAY' , IFADD , IFIND , 'IRAY' ,                   *
C  *                 ISAVD , J(L) , JSTART , 'LABEL' , LABELT(L) ,       *
C  *                 NNODES                                              *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6000
6000  FORMAT(" YOU ARE NOW MODIFYING THE TREE.",
     2       " ENTER A LABEL OF 'DONE' OR '' TO STOP.")
C
100   CONTINUE
      IF(IPRINT.GT.0)CALL PRNT
      WRITE 6001
6001  FORMAT(1X,6HLABEL?)
      READ 5001,LABELT
5001  FORMAT(A10)
      IF(LABELT.EQ." ".OR.LABELT.EQ."DONE")GO TO 999
C       THEN USER IS FINISHED
C       ELSE GET THE NRN VECTOR
          CALL NODIN
C
      IF(NLVLS.LE.0)GO TO 999
C       THEN USER OPTED OUT WHEN INPUTTING NRN VECTOR
C       ELSE ADD NODE(S) AND LABEL AS REQUIRED
C
          ISAVD = 0
C
          IF(NDIFF.GT.0)GO TO 200
C           THEN NEED TO ADD AT LEAST ONE NODE
C           ELSE CHANGE LABEL ONLY
              CALL LSET(IFIND,LABELT)
              GO TO 100
C
200       CONTINUE
          JSTART = NLVLS - NDIFF + 1
```

```
C
C     ********************* ADD ALL NEW NODES *********************
C
          DO 300 J = JSTART,NLVLS
            NNODES = NNODES + 1
            CALL ISET(IFIND,IFADD,NNODES)
            CALL LSET(NNODES,BLANK)
            CALL ISET(NNODES,1,INNRN(J))
            CALL ISET(NNODES,2,-1)
            CALL ISET(NNODES,3,-1)
            CALL ISET(NNODES,4,IFIND)
            CALL ASET(NNODES,1,0.0)
            CALL ASET(NNODES,2,0.0)
            IFIND = NNODES
            IFADD = 2
300       CONTINUE
C
C         LABEL APPLIES ONLY TO LAST NODE IN BRANCH ADDED
C
          CALL LSET(NNODES,LABELT)
          GO TO 100
999   CONTINUE
      RETURN
      END
```

```
CPAGE START ARAY   (ACTS AS AN ARRAY ... W = ARAY(I,J))
      FUNCTION ARAY(I,J)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
C  ********************************************************************
C  *                                                                  *
C  *  FUNCTION ARAY(I,J)                                              *
C  *  USE: TO ACCESS THE WEIGHTS ASSOCIATED WITH A NODE.              *
C  *       IT IS TREATED AS AN ARRAY (WHAT I'VE CALLED                *
C  *       A PSUEDO-ARRAY).  'ARAY'(*,1) IS THE INPUT                 *
C  *       WEIGHT FOR NODE(*) (NORMALIZED), AND 'ARAY'(*,2)           *
C  *       IS THE CUMULATIVE WEIGHT (CUMWT) FOR NODE(*).              *
C  *       THE VALUE OF 'ARAY'(I,J) IS SET TO V BY USE OF             *
C  *       THE ROUTINE VSET (CALL VSET(I,J,V)).                       *
C  *  CALLED BY: CALC , COPYR , DSPLAY , DSPLOT ,                     *
C  *            NUMREV , PRNT , SENSTV , SNSPLT                       *
C  *  ROUTINES CALLED: READMS(S)                                      *
C  *  VARIABLES:                                                      *
C  *       USED: I(P) , J(P) , NTAPE                                  *
C  *       MODIFIED: ARRAY , ISIN                                     *
C  *                                                                  *
C  ********************************************************************
C
      IF(I.NE.ISIN)CALL READMS(NTAPE,ARRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      ARAY = ARRAY(J+4)
      ISIN = I
      RETURN
      END
CPAGE START ASET   (SETS PSUEDO-ARRAY ARAY(I,J) = V )
      SUBROUTINE ASET(I,J,V)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE ASET(I,J,V)                                          *
C  *  USE: TO SET THE VALUE OF THE PSUEDO-ARRAY 'ARAY' TO             *
C  *       THE INPUT VALUE V.                                         *
C  *  CALLED BY: ADD , CALC , COPYR , RDWT , SPAN                     *
C  *  ROUTINES CALLED: READMS(S) , WRITMS(S)                          *
C  *  VARIABLES:                                                      *
C  *       USED: I(P) , J(P) , NTAPE , V(P)                           *
C  *       MODIFIED: 'ARAY' , ARRAY , ISIN                            *
C  *                                                                  *
C  ********************************************************************
C
      IF(I.NE.ISIN)CALL READMS(NTAPE,ARRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      ARRAY(J+4) = V
C
C     SAVE THE NEW DATA ON DISK FILE
      CALL WRITMS(NTAPE,ARRAY,64,I,1)
      ISIN = I
      RETURN
      END
```

```
CPAGE START CALC   (CALCULATES TREE VALUES FOR NON-DATA NODES)
      SUBROUTINE CALC
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SYS/NSYS,LSYS(63)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE CALC                                                    *
C  *  USE: TO CALCULATE THE INNER (NON-DATA NODE) VALUES                 *
C  *       FOR ALL SYSTEMS.  SHOULD BE USED WHENEVER                     *
C  *       WEIGHTS OR VALUES CHANGE.  (COLLAPSES TREE)                   *
C  *  CALLED BY: PRUNE , WVC                                             *
C  *  ROUTINES CALLED: NEXT , PRETOT                                     *
C  *  VARIABLES:                                                         *
C  *       USED: ICONT , IDATA , IFIND , LEVEL ,                         *
C  *             LVL , NNODES , NSYS                                     *
C  *       MODIFIED: 'ARAY' , I(L) ,                                     *
C  *                 ISTOP(L) , J(L) , NDEEP , VHOLD(L) , 'VRAY'       *
C  *                                                                     *
C  ***********************************************************************
C
      IF(NNODES.LE.0)GO TO 999
      WRITE 6001
6001  FORMAT(" INTERIOR TREE VALUES BEING CALCULATED.")
      NDEEP = 0
      CALL PRETOT
      CALL ASET(1,1,1.0)
      CALL ASET(1,2,1.0)
      CALL ASET(2,1,1.0)
      CALL ASET(2,2,1.0)
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 999
      IF(LVL.GT.NDEEP)NDEEP = LVL
C       THEN UPDATE THE DEPTH COUNTER (NDEEP)
C     CALCULATE THE CUMULATIVE WEIGHT FOR THIS NODE
      VHOLD = ARAY(IFIND,1)
      IF((LVL-1).GE.1)
     2  VHOLD = ARAY(IFIND,1)*ARAY(LEVEL(LVL-1,1),2)
      CALL ASET(IFIND,2,VHOLD)
      IF(IDATA.EQ.1)GO TO 300
C       THEN THIS IS A DATA NODE, SO MUST ADD IN ITS CONTRIBUTION
C       ELSE IT IS AN INTERNAL NODE AND MUST CLEAR IT
          DO 200 J = 1 , NSYS
            CALL VSET(IFIND,J,0.0)
200         CONTINUE
      GO TO 500
300   CONTINUE
C
C     ADD IN THE CONTRIBUTION OF THIS DATA NODE
      ISTOP = LVL - 1
```

```
      DO 400 I = 1 , ISTOP
      DO 400 J = 1 , NSYS
      VHOLD = VRAY(LEVEL(I,1),J)
     2 + ARAY(IFIND,2)*VRAY(IFIND,J)
      CALL VSET(LEVEL(I,1),J,VHOLD)
400   CONTINUE
500   CONTINUE
      CALL NEXT
      GO TO 100
999   CONTINUE
      RETURN
      END
```

```
CPAGE START CONTRL (MENU CONTROLLED MODIFICATION OF CONTROL VARIABLES)
      SUBROUTINE CONTRL
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      DIMENSION ITEM(1)
      EQUIVALENCE (ITEM(1),IPRINT)
      DIMENSION VARBLS(10)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE CONTRL                                                  *
C  *  USE: TO MODIFY CERTAIN CONTRL VARIABLES, ESPECIALLY                *
C  *       THE PRINT CONTROL FLAG (IPRINT).  CHANGING                    *
C  *       NTAPE WILL ONLY RESULT IN AN ERROR CONDITION,                 *
C  *       AND CHANGING MODE WILL HAVE NO EFFECT, AS IT IS               *
C  *       NOT USED AT THIS TIME.                                        *
C  *  CALLED BY: DRIVER                                                  *
C  *  ROUTINES CALLED: MENU                                              *
C  *  VARIABLES:                                                         *
C  *       USED: IWANT(L) , VARBLS(L)                                    *
C  *       MODIFIED: ITEM(EQUIVALENCED TO PRINT, ISAVD, NTAPE            *
C  *                 AND MODE)                                           *
C  *                                                                     *
C  ***********************************************************************
C
      DATA VARBLS/"IPRINT","ISAVD","NTAPE","MODE",6*" "/
      WRITE 6000
6000  FORMAT(" SELECT ... CONTROL VARIABLE TO MODIFY.")
      CALL MENU(3,VARBLS,IWANT)
      IF(IWANT.EQ.0)GO TO 999
      WRITE 6001,VARBLS(IWANT),ITEM(IWANT)
6001  FORMAT(1X,"OLD VALUE OF ",A6," WAS ",I2," , NEW VALUE IS?")
      READ *,ITEM(IWANT)
999   CONTINUE
      RETURN
      END
```

```
CPAGE START COPYR  (COPIES NODE TO NODE)
      SUBROUTINE COPYR
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
      LOGICAL YES,NO
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE COPYR                                                   *
C  *  USE: TO COPY ALL OF A DATA STRUCTURE BELOW A NODE                  *
C  *       TO A SECOND NODE (WHICH MUST ALREADY EXIST).                  *
C  *       THIS IS A HANDY WAY TO CREATE LARGE, SYMMETRIC                *
C  *       DATA STRUCTURES.                                              *
C  *  CALLED BY: DRIVER                                                  *
C  *  ROUTINES CALLED: NEXT , NODIN , PRENEX                             *
C  *  VARIABLES:                                                         *
C  *       USED: ICONT , IDATA , IFADD , IFIND , LVL ,                   *
C  *             NDIFF , NLVLS                                           *
C  *       MODIFIED: 'ARAY' , IFINDT(L) , 'IRAY' , 'LABEL' ,             *
C  *                 LEVEL , NNODES                                      *
C  *                                                                     *
C  ***********************************************************************
C
C     ************  GET AND CHECK NODE TO WHICH TO COPY  ************
C
C     EXIT IF (USER OPTS OUT OR NODE DOESN'T EXIST)
C
100   CONTINUE
      WRITE 6000
6000  FORMAT(" ENTER NODE TO BE COPIED TO.")
      CALL NODIN
      IF(NLVLS.LE.0)GO TO 999
C       THEN OPTED OUT ELSE
      IF(NDIFF.LE.0)GO TO 200
C       THEN NODE WAS FOUND
C       ELSE
          WRITE 6001
6001      FORMAT(" NODE MUST EXIST BEFORE COPYING IS ALLOWED.")
          GO TO 999
C
C     ************ GET AND CHECK NODE FROM WHICH TO COPY ************
C
C     EXIT IF (USER OPTS OUT OR NODE DOESN'T EXIST OR
C              (NODE EXISTS BUT IS A DATA NODE (NOTHING TO COPY))
C
200   CONTINUE
      IFINDT = IFIND
      WRITE 6002
6002  FORMAT(" ENTER NODE TO BE COPIED.")
      CALL PRENEX
      IF(ICONT.NE.0)GO TO 250
```

```
C       THEN NODE TO BE COPIED WAS FOUND
C       ELSE NODE WAS NOT FOUND, OR USER OPTED OUT
          IF(NDIFF.GT.0)WRITE 6004
6004      FORMAT(" NODE TO BE COPIED DOESN'T EXIST.")
          GO TO 999
250   CONTINUE
      LEVEL(LVL,3) = IFINDT
      IF(IDATA.EQ.0)GO TO 300
C       THEN EVERYTHING IS GO FOR COPY
C       ELSE THERE IS NOTHING TO COPY (DATA NODE)
          WRITE 6003
6003      FORMAT(" NODE ENTERED IS A DATA NODE, ",
     2           "AND CANNOT BE COPIED FROM.")
          GO TO 999
C
C     ***************** BEGIN COPYING OPERATION *****************
300   CONTINUE
      CALL NEXT
      IF(ICONT.EQ.0)GO TO 400
      IFINDT = LEVEL(LVL,3)
      IF(IFADD.EQ.2)IFINDT = LEVEL(LVL-1,3)
      NNODES = NNODES + 1
      CALL ISET(IFINDT,IFADD,NNODES)
      CALL LSET(NNODES,LABEL(IFIND))
      CALL ISET(NNODES,1,IRAY(LEVEL(LVL,1),1))
      CALL ISET(NNODES,2,-1)
      CALL ISET(NNODES,3,-1)
      CALL ISET(NNODES,4,IFINDT)
      CALL ASET(NNODES,1,ARAY(IFIND,1))
      CALL ASET(NNODES,2,ARAY(IFIND,2))
      LEVEL(LVL,3) = NNODES
      GO TO 300
400   CONTINUE
      WRITE 6005
6005  FORMAT(" COPYING COMPLETED.")
C
999   CONTINUE
      RETURN
      END
```

```
CPAGE START CRAY   (ACTS AS AN ARRAY ... "COMMENTS" = CRAY(I))
      SUBROUTINE CRAY(I)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
      COMMON/SYS/NSYS,LSYS(63)
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE CRAY(I)                                              *
C  *  USE: TO ACCESS THE COMMENTS WHICH ARE ELICITED                  *
C  *       WHEN INPUTTING WEIGHTS OR VALUES (RATIONALE                *
C  *       CAPTURING DEVICE).                                         *
C  *       SEE THE FUNCTION ARAY FOR A COMPLETE DISCUSSION.           *
C  *  CALLED BY: REASON , SUMMRY                                      *
C  *  ROUTINES CALLED: READMS(S)                                      *
C  *  VARIABLES:                                                      *
C  *       USED: I(P) , NSYS , NTAPE                                  *
C  *       MODIFIED: ARRAY , ISIN , ISTART(L) ,                       *
C  *                 ISTOP(L) , J(L)                                  *
C  *                                                                  *
C  ********************************************************************
C
      ISTART = 8 + NSYS
      IF(ISIN.NE.I)CALL READMS(NTAPE,ARRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
1     CONTINUE
      IF(ARRAY(ISTART).EQ." ")GO TO 999
      ISTOP = ISTART + 7
      IF(ISTOP.GT.64)GO TO 999
      WRITE 6001,(ARRAY(J),J=ISTART,ISTOP)
6001  FORMAT(1X,8A10)
      ISTART = ISTART + 8
      GO TO 1
999   CONTINUE
      ISIN = I
      RETURN
      END
CPAGE START CSET   (SETS PSUEDO-ARRAY CRAY(I) = "COMMENTS")
      SUBROUTINE CSET(I)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
      COMMON/SYS/NSYS,LSYS(63)
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE CSET(I)                                              *
C  *  USE: TO SET THE VALUE OF THE PSUEDO-ARRAY 'CRAY' TO             *
C  *       THE INPUT COMMENTS (READS USER'S INPUT                     *
C  *       RATIONALES AND STORES THEM).                               *
C  *  CALLED BY: RDTTL , RDV , RDWT                                   *
C  *  ROUTINES CALLED: READMS(S) , WRITMS(S)                          *
C  *  VARIABLES:                                                      *
C  *       USED: I(P) , NSYS , NTAPE                                  *
C  *       MODIFIED: ARRAY , 'CRAY' , ISIN , ISTART(L) ,              *
```

```
C  *                ISTOP(L) , J(L)                                    *
C  *                                                                   *
C  *********************************************************************
C
      ISTART = 8 + NSYS
      IF(ISIN.NE.I)CALL READMS(NTAPE,ARRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
100   CONTINUE
      ISTOP = ISTART + 7
      IF(ISTOP.GT.64)GO TO 200
      WRITE 6001
6001  FORMAT(2H ?)
      READ 5001,(ARRAY(J),J=ISTART,ISTOP)
5001  FORMAT(8A10)
      IF(ARRAY(ISTART).EQ." ")GO TO 200
      ISTART = ISTART + 8
      GO TO 100
200   CONTINUE
C
C     SAVE THE NEW DATA ON DISK FILE
      CALL WRITMS(NTAPE,ARRAY,64,I,1)
      ISIN = I
      RETURN
      END
```

```
CPAGE START DISPLA (DRIVER FOR SINGLE NODE DISPLAYS AND PLOTS)
      SUBROUTINE DISPLA
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE DISPLA                                               *
C  *  USE: TO DRIVE ROUTINES DSPLAY AND DSPLOT TO PRODUCE             *
C  *       DISPLAYS OF NODES.  THIS ROUTINE WILL DISPLAY              *
C  *       THE DATA ASSOCIATED WITH ONE NODE AT A TIME.               *
C  *  CALLED BY: DRIVER                                               *
C  *  ROUTINES CALLED: DSPLAY , DSPLOT , NODIN                        *
C  *  VARIABLES:                                                      *
C  *       USED: NDIFF , NLVLS                                        *
C  *       MODIFIED: (NONE)                                           *
C  *                                                                  *
C  ********************************************************************
C
1     CONTINUE
      WRITE 6001
6001  FORMAT(" ENTER NODE TO BE DISPLAYED.")
      CALL NODIN
      IF(NLVLS.LE.0.OR.NDIFF.GT.0)GO TO 2
C       THEN DONE
C       ELSE
          CALL DSPLAY
          CALL DSPLOT
          GO TO 1
2     CONTINUE
      RETURN
      END
```

```
CPAGE START DRIVER (MAIN DRIVER, USES SELECTION BY NAME, NOT NUMBER)
      SUBROUTINE DRIVER
      LOGICAL YES,NO
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      DIMENSION OPT(20)
      DATA NOPT/20/
      DATA OPT/3HSEL,3HNEW,3HMOD,3HATT,3HSYS,3HCOP,3HPRU,
     2 3HWVC,3HDIS,3HSUM,3HSEN,3HRAT,3HWOR,3HDON,3HHEL,3HCON,
     3 3HREV,3HSPA,3HNUM,3HTTL/
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE DRIVER                                                  *
C  *  USE: AS MAIN DRIVER FOR SELECTION OF OPTIONS.                      *
C  *       OPTIONS ARE SELECTED BY NAME INPUT (FIRST                     *
C  *       THREE (3) CHARACTERS ONLY).  THIS ROUTINE                     *
C  *       WRITES OUT A MENU IF REQUESTED, OR IF THE                     *
C  *       USER DISPLAYS IGNORANCE OF THE OPTIONS                        *
C  *       (BY FAILIING TO SELECT A VALID OPTION WITHIN                  *
C  *       THREE ATTEMPTS).                                              *
C  *  CALLED BY: DASS                                                    *
C  *  ROUTINES CALLED: ADD , CONTRL , COPYR , DISPLA ,                   *
C  *                   INITT , NO , NUMREV , PRUNE ,                     *
C  *                   RDATT , RDSYS , RDTTL , REASON ,                  *
C  *                   REVIEW , SENSTV , SPAN , SUMMRY ,                 *
C  *                   TAPER , TLOAD , TSAVE , WVLOAD ,                  *
C  *                   WRKSHT                                            *
C  *  VARIABLES:                                                         *
C  *       USED: NNODES , NOPT(L) , OPT(L)                               *
C  *       MODIFIED: I(L) , IPRINT , IWANT(L) , NERR(L) ,                *
C  *                 NTAPE , USERIN(L)                                   *
C  *                                                                     *
C  ***********************************************************************
C
6000  FORMAT(1H1,/,3X,
C
     2 "YOUR OPTIONS ARE AS FOLLOWS ..."                          ,/,/,3X,
C
     3 "INPUT (ROUTINE USED)  FUNCTION PERFORMED"                 ,/,/,3X,
C
     4 "ATT (RDATT ) ATTRIBUTE LABEL ENTRY (REGRET/VALUE)"        ,/,/,3X,
C
     5 "CON (CONTRL) CONTROL FLAG CHANGES (IPRINT,ISAVD,NTAPE)",/,/,3X,
C
     6 "COP (COPYR ) COPIES ONE NODE TO ANOTHER (INCLUDING"        ,/,3X,
     7 "             ALL DOWN BRANCHES)"                           ,/)
C
6001  FORMAT(3X,
C
     2 "DIS (DISPLA) DISPLAY ONE NODE"                            ,/,/,3X,
C
     3 "DON (      ) DONE WITH WORK, CLOSE LAST FILE"             ,/,/,3X,
C
```

```
     4 "REL (      ) WILL IMMEDIATELY ALLOW REPRINT"            ,/,3X,
     5 "              OF THIS MENU"                            ,/,/,3X,
C
     6 "MOD (ADD   ) MODIFIES EXISTING TREE, NODE BY NODE"     ,/,/,3X,
C
     7 "NEW (INITT ) NEW TREE BUILDING DRIVER"                 ,/,/,3X,
C
     8 "NUM (NUMREV) SAME AS REVIEW, BUT WITH WEIGHTS"          ,/,3X,
     9 "              AND VALUES THROWN IN."                   ,/,/,3X,
C
     X "PRU (PRUNE ) PRUNES THE TREE NODES"                    ,/,/,3X,
C
     A "RAT (REASON) RATIONALE PRINTING"                       ,/,/,3X,
C
     B "REV (REVIEW) REVIEW PRINT OF TREE"                     ,/,/,3X,
C
     C "SEL (TAPER ) SELECTS TREE FILE AND OPENS OR"            ,/,3X,
     D "              CLOSES FILES AS NECESSARY."              ,/,/,3X,
C
     E "SEN (SENSTV) SENSITIVITY ON WEIGHT OF A BRANCH"        ,/)
6002  FORMAT(3X,
C
     2 "SPA (SPAN  ) ADD TO TREE BY ADDING DOWNLINK"            ,/,3X,
     3 "              NODES TO CURRENT NODES"                  ,/,/,3X,
C
     4 "SUM (SUMMRY) SUMMARY DISPLAYS OF TREE OR NODE AND"      ,/,3X,
     5 "              ITS BRANCHES (MULTIPLE DISPLAYS)"        ,/,/,3X,
C
     6 "SYS (RDSYS ) SYSTEM LABEL ENTRY"                       ,/,/,3X,
C
     7 "TTL (RDTTL ) DATA FILE TITLE ENTRY"                    ,/,/,3X,
C
     8 "WOR (WRKSHT) WORKSHEET GENERATOR"                      ,/,/,3X,
C
     9 "WVC (WVLOAD) DRIVER FOR LOADING WEIGHTS AND VALUES,"    ,/,3X,
     A "              AND RECALCULATING TREE VALUES."          ,/,/,3X,
C
     B "      ANY OTHER ENTRY IS IGNORED (UP TO THREE TRIES)."  ,/)
C
C
      IPRINT = 0
      NTAPE = 1
      CALL TLOAD
      IF(NNODES.GT.0)GO TO  50
C       THEN TLOAD OPENED A FILE WITH A TREE STRUCTURE
C       ELSE MIGHT AS WELL SAVE USER ONE INPUT, AND GO TO OPTION 'NEW'
          WRITE 6003
6003      FORMAT(" FILE HAS NO TREE, PROGRAM GOING TO OPTION 'NEW'.")
          CALL INITT
50    CONTINUE
100   CONTINUE
      NERR = 0
      IF(NO(3,"DO YOU NEED TO SEE THE MENU?"))GO TO 200
150   CONTINUE
```

```
      WRITE 6000
      WRITE 6001
      WRITE 6002
200   CONTINUE
      WRITE 6004
6004  FORMAT(" OPTION?")
      READ 5001,USERIN
5001  FORMAT(A3)
      DO 300 I = 1 , NOPT
      IWANT = I
      IF(OPT(IWANT).EQ.USERIN)GO TO 400
300   CONTINUE
      NERR = NERR + 1
      IF(NERR.GE.3)GO TO 100
      GO TO 200
400   CONTINUE
C
C     TO GET HERE, THE USER MUST BE CLEAR ON WHAT TO DO, SO
C     SET ERROR COUNTER TO NONE (0)
C
      NERR = 0
      GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,100,16,17,18,19,20),IWANT
1     CALL TAPER
      GO TO 200
2     CALL INITT
      GO TO 200
3     CALL ADD
      GO TO 200
4     CALL RDATT
      GO TO 200
5     CALL RDSYS
      GO TO 200
6     CALL COPYR
      GO TO 200
7     CALL PRUNE
      GO TO 200
8     CALL WVLOAD
      GO TO 200
9     CALL DISPLA
      GO TO 200
10    CALL SUMMRY
      GO TO 200
11    CALL SENSTV
      GO TO 200
12    CALL REASON
      GO TO 200
13    CALL WRKSHT
      GO TO 200
16    CALL CONTRL
      GO TO 200
17    CALL REVIEW
      GO TO 200
18    CALL SPAN
      GO TO 200
```

```
19    CALL NUMREV
      GO TO 200
20    CALL RDTTL
      GO TO 200
14    CALL TSAVE
999   CONTINUE
      STOP "END OF PROGRAM RUN"
      END
```

```
CPAGE START DSPLAY (GENERATES A SPECIAL FORMAT NODE DISPLAY)
      SUBROUTINE DSPLAY
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SYS/NSYS,LSYS(63)
      DIMENSION TEMP(10)
      DATA BLANK/1H / , CUMWT/5HCUMWT/ , STAR/1H*/
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE DSPLAY                                               *
C  *  USE: GENERATES A TABULAR DISPLAY OF THE DATA                    *
C  *       ASSOCIATED WITH THE SELECTED NODE.  ALL NUMBERS            *
C  *       ARE NORMALIZED TO 100.0 (VALUES) OR 1.0 (WEIGHTS).         *
C  *  CALLED BY: DISPLA , RDWT , SUMMRY                               *
C  *  ROUTINES CALLED: (NONE)                                         *
C  *  VARIABLES:                                                      *
C  *       USED: 'ARAY' , BLANK(L) , CUMWT(L) , IDATA ,               *
C  *             IFIND , 'IRAY' , 'LABEL' , LEVEL , LSYS ,            *
C  *             LVL , NSYS , STAR(L) , 'VRAY'                        *
C  * ,     MODIFIED: DUM(L) , I(L) , IFINDT(L) , J(L) ,               *
C  *                 WT(L)                                            *
C  *                                                                  *
C  ********************************************************************
C
C        DISPLAY FORMAT IS AS FOLLOWS ...
C
C          1 MAIN      -PRICE     -
C             FACTOR    WT      SYS1      SYS2      CUMWT
C            1)FACTOR 1  (.33)  100.00    100.00     .33
C            2)FACTOR 2 *(.67)   45.00     78.00     .67
C               TOTAL             65.98     84.00    1.00
C
C  ********************************************************************
      IF(IDATA.EQ.0)GO TO 2
C       THEN IFIND IS NOT A DATA NODE, AND CAN BE DISPLAYED
C       ELSE
        WRITE 6000
6000    FORMAT(" DSPLAY ... NODE IS A DATA NODE AND ",
     2          "CANNOT BE DISPLAYED.")
        GO TO 999
2     CONTINUE
C     THE FORMATS 6001,6002,6003 AND 6004 ARE SET FOR
C       132 COLUMNS OF OUTPUT (ROUTINE DSPLAY)
      WRITE 6001,LVL,(IRAY(LEVEL(I,1),1),I=1,LVL),
     2 (LABEL(LEVEL(J,1)),J=1,LVL)
6001  FORMAT(/,/,1X,=(I2,1X),5(/,1X,10(A10,"-")))
      WRITE 6002,(LSYS(I),I=1,NSYS),CUMWT
6002  FORMAT(5X,"FACTOR",7X,"WT    ",10(A5,2X),
     2 5(/,24X,10(A5,2X)))
      IFINDT = IRAY(IFIND,2)
100   CONTINUE
```

```
      IF(IFINDT.LE.0)GO TO 400
C       THEN DONE WITH ALL DESCENDENTS
C       ELSE DISPLAY THIS DESCENDENT
C
C   ********************* DISPLAY THIS NODE *************************
C
C     ASSUME THAT THIS BRANCH IS A DATA BRANCH
      DUM = STAR
      WT = 1.0
      IF(IRAY(IFINDT,2).LE.0)GO TO 200
C       THEN IT IS A DATA BRANCH
C       ELSE IT IS NOT A DATA BRANCH
          DUM = BLANK
          WT = ARAY(IFINDT,2)
200   CONTINUE
      WRITE 6003,IRAY(IFINDT,1),LABEL(IFINDT),DUM,ARAY(IFINDT,1),
     2 ((VRAY(IFINDT,J)/WT),J=1,NSYS),ARAY(IFINDT,2)
6003  FORMAT(1X,I2,")",A10,A1,"  (",F3.2,")",10F7.2,
     2  5(/,22X,10F7.2))
C     GET NEXT CROSSLINK
      IFINDT = IRAY(IFINDT,3)
      GO TO 100
400   CONTINUE
C     WRITE OUT SUMMARY OF THIS NODE
      WRITE 6004,((VRAY(IFIND,J)/ARAY(IFIND,2)),J=1,NSYS),
     2 ARAY(IFIND,2)
6004  FORMAT(5X,"TOTAL",12X,10F7.2,5(/,22X,10F7.2))
      WRITE 6005
6005  FORMAT(/,/)
999   CONTINUE
      RETURN
      END
```

```
CPAGE START DSPLOT (GENERATES A SPECIAL FORMAT NODE PLOT)
      SUBROUTINE DSPLOT
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SYS/NSYS,LSYS(63)
      DIMENSION ALFA(20),DSPL(51)
      DATA ALFA/1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,
     2          1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT/
      DATA RSCALE/2.0/
      DATA VERT/1H+/
      DATA BLANK/1H / , CUMWT/5HCUMWT/ , STAR/1H*/
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE DSPLOT                                               *
C  *  USE: GENERATES A PLOTTED DISPLAY OF THE DATA                    *
C  *       ASSOCIATED WITH THE SELECTED NODE (SIMILAR                 *
C  *       INFORMATION TO THAT IN THE DSPLAY FORMAT                   *
C  *       BUT IN PLOTTED RATHER THAN TABULAR FORM).                  *
C  *       IT USES FORMATTED WRITES TO PLOT THE INFORMATION,          *
C  *       THUS ALTHOUGH IT DOES NOT PRESENT A PRECISE                *
C  *       PICTURE (GRID IS N BY 51 SPACES, RESOLUTION                *
C  *       IS +/- 2 UNITS), IT IS NOT TIED TO A PARTICULAR            *
C  *       PLOT SYSTEM.                                               *
C  *  CALLED BY: DISPLA , SUMMRY                                      *
C  *  ROUTINES CALLED: (NONE)                                         *
C  *  VARIABLES:                                                      *
C  *       USED: ALFA(L) , 'ARAY' , BLANK(L) , IDATA ,                *
C  *             IFIND , 'IRAY' , 'LABEL' , NSYS ,                    *
C  *             RSCALE(L) , STAR(L) , VERT(L) , 'VRAY'               *
C  *       MODIFIED: DSPL(L) , DUM(L) , I(L) , IFINDT(L) ,            *
C  *                 IFROM(L) , IPUT(L) , J(L) , WT(L)                *
C  *                                                                  *
C  ********************************************************************
C
C         PLOT FORMAT IS AS FOLLOWS ...
C
C          BRANCH       0--------20--------40--------60-------100
C          DWNLINK 1    +          +  C   D  +  A       B         +
C          DWNLINK 2 *  +          A   B     D+     C    +         +
C          DWNLINK 3    +   B   C  +          +          +         +
C          BRANCH       0--------20--------40--------60-------100
C
C  ********************************************************************
C
      IF(IDATA.EQ.0)GO TO 2
C       THEN IFIND IS NOT A DATA NODE, AND CAN BE DISPLAYED
C       ELSE
        WRITE 6000
6000    FORMAT(" DSPLOT ... NODE IS A DATA NODE AND ",
     2         "CANNOT BE PLOTTED.")
        GO TO 999
```

```
C
2     CONTINUE
C
C     THE FORMATS 6000,6001,6002 AND 6003 ARE SET TO FIT WITHIN 80 COLS
C
      WRITE 6001
6001  FORMAT(/)
      WRITE 6002
6002  FORMAT(" BRANCH      ",
     2 " 0--------20--------40--------60--------80-------100")
      IFROM = 2
      IFINDT = IFIND
100   CONTINUE
      IFINDT = IRAY(IFINDT,IFROM)
      IFROM = 3
      IF(IFINDT.LE.0)GO TO 500
C       THEN DONE WITH ALL THE DESCENDENTS
C       ELSE PLOT THIS DESCENDENT
      DO 200 I = 2 , 50
      DSPL(I) = BLANK
200   CONTINUE
      DO 201 I = 1 , 51 , 10
      DSPL(I) = VERT
201   CONTINUE
C
C     ASSUME THAT THIS DESCENDENT IS A DATA BRANCH
      DUM = STAR
      WT = 1.0
      IF(IRAY(IFINDT,2).LE.0)GO TO 300
C       THEN THIS BRANCH IS A DATA BRANCH
C       ELSE IT IS NOT A DATA BRANCH
          DUM = BLANK
          WT = ARAY(IFINDT,2)
300   CONTINUE
      DO 400 J = 1 , NSYS
      IPUT = (VRAY(IFINDT,J)/WT)/RSCALE + 1.9
      DSPL(IPUT) = ALFA(J)
400   CONTINUE
      WRITE 6003,LABEL(IFINDT),DUM,DSPL
6003  FORMAT(1X,A10,A1,2X,51A1)
      GO TO 100
500   CONTINUE
      WRITE 6002
      WRITE 6001
999   CONTINUE
      RETURN
      END
```

```
CPAGE START DVIDE  (DIVIDES ELEMENTS OF A VECTOR BY A CONSTANT)
      SUBROUTINE DVIDE(N,A,D)
      DIMENSION A(N)
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE DVIDE(N,A,D)                                             *
C  *  USE: TO DIVIDE ALL ELEMENTS OF THE INPUT VECTOR (A)                 *
C  *       BY THE DIVISOR (D).  THIS IS USED IN                           *
C  *       NORMALIZING THE INPUT WEIGHTS FOR THE                          *
C  *       BRANCHING NODES.                                               *
C  *  CALLED BY: RDWT                                                     *
C  *  ROUTINES CALLED: (NONE)                                             *
C  *  VARIABLES:                                                          *
C  *       USED: D(P) , N(P)                                              *
C  *       MODIFIED: A(P) , I(L)                                          *
C  *                                                                      *
C  ************************************************************************
C
      DO 100 I = 1 , N
      A(I) = A(I)/D
100   CONTINUE
      RETURN
      END
```

```
CPAGE START FIND   (ROUTINE TO FIND MATCH TO NODE IN INNRN)
      SUBROUTINE FIND
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE FIND                                                    *
C  *  USE: TO FIND THE NODE WHICH MATCHES THE NRN VECTOR                 *
C  *       FOUND IN THE ARRAY INNRN (LOADED BY                           *
C  *       ROUTINE NODIN).  IF THERE IS NO MATCH, THE                    *
C  *       APPROPRIATE FLAGS ARE SET (IFADD, NDIFF) AND                  *
C  *       THE FOUND NODE IS THE LAST NODE WHICH                         *
C  *       EXISTS WHICH WOULD LIE ON THE BRANCH TO THE                   *
C  *       NODE SPECIFIED IN THE ARRAY INNRN.                            *
C  *  CALLED BY: NODIN                                                   *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: INNRN , 'IRAY' , NLVLS                                  *
C  *       MODIFIED: IDATA , IFADD , IFIND , J(L) ,                      *
C  *                 LEVEL , LVL , NDIFF                                 *
C  *                                                                     *
C  ***********************************************************************
C
C         INPUT  NLVLS (THE LENGTH OF THE INNRN VECTOR)
C                INNRN (THE VECTOR OF NRN NUMBERS)
C         OUTPUT IFIND (THE CLOSEST MATCHING NODE)
C                NDIFF (THE NUMBER OF LEVELS WHICH WERE NOT
C                       MATCHED (IN INNRN))
C                IFADD (2 IF THE INNRN NODE IS DESCENDENT TO
C                         THE IFIND NODE; 3 IF THE INNRN NODE
C                         IS BROTHER TO THE IFIND NODE (USED
C                         IF THE NEW NODE IS TO BE ADDED))
C                LVL   (THE NUMBER OF LEVELS INTO TREE TO
C                       REACH IFIND)
C                LEVEL ((I,1) CONTAINS THE LOCATION OF THE ITH
C                       NODE ALONG THE BRANCH TO IFIND
C                       (I,2) IS USED TO FLAG THE STATUS OF THE
C                       ITH NODE)
C
C     *************** PREPARE TO START ****************
C
      IFIND = 1
      NDIFF = 0
      IFADD = 2
C
C     ********************** SEARCH **********************
C
      DO 400 J = 1 , NLVLS
C
100   CONTINUE
C
      IF(INNRN(J).EQ.IRAY(IFIND,1))GO TO 300
      IF(IRAY(IFIND,3).GT.0)GO TO 200
```

```
C       ELSE NO MORE CROSSLINKING NODES EXIST
C            AND NO MATCHING CROSSLINKED NODE
C            MEANS THAT THE NEW NODE, IF ADDED,
C            WILL BE CROSSLINKED TO THE NODE(IFIND)
          IFADD = 3
          GO TO 500
C
200   CONTINUE
      IFIND = IRAY(IFIND,3)
      GO TO 100
C
300   CONTINUE
C     ADD A LEVEL TO MATCHING NODES LIST (LEVEL)
      LVL = J
      LEVEL(LVL,1) = IFIND
      LEVEL(LVL,2) = 0
      IF(J.EQ.NLVLS)GO TO 999
C       THEN WE HAVE A COMPLETE MATCH
C       ELSE NEED TO CHECK FOR MATCH AT NEXT LEVEL DOWN (DEPTH)
          IF(IRAY(IFIND,2).LE.0)GO TO 600
C           THEN THERE IS NO MORE DEPTH TO THE TREE
C           ELSE THERE IS DOWN LINK SO CHECK ON
              LEVEL(LVL,2) = 1
              IFIND = IRAY(IFIND,2)
400           CONTINUE
C
          WRITE 6000
6000      FORMAT(" FIND ... THIS LINE SHOULD NEVER EXECUTE.")
          STOP "ERROR IN FIND"
C
500       CONTINUE
C
C     *** HERE WE WERE UNABLE TO MATCH THE J-TH LEVEL ****
C
          NDIFF = NLVLS - J + 1
          GO TO 999
C
600   CONTINUE
C
C     ** HERE WE WERE UNABLE TO MATCH THE (J-1)TH LEVEL **
C
      NDIFF = NLVLS - J
C
999   CONTINUE
      IDATA = 0
      IF(IRAY(IFIND,2).LE.0)IDATA = 1
      RETURN
      END
```

```
CPAGE START INITT  (NEW TREE BUILDER, CALLS RDATT,RDSYS,RDTTL,SPAN)
      SUBROUTINE INITT
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE INITT                                                   *
C  *  USE: DRIVER FOR BUILDING A NEW TREE, IT AUTOMATICALLY              *
C  *       CALLS THE STRUCTURE BUILDING ROUTINES NECESSARY               *
C  *       TO BUILD A TREE STRUCTURE.                                    *
C  *  CALLED BY: DRIVER , TAPER                                          *
C  *  ROUTINES CALLED:                                                   *
C  *  ROUTINES CALLED: RDATT , RDSYS , RDTTL , SPAN                      *
C  *  VARIABLES:                                                         *
C  *       USED: (NONE)                                                  *
C  *       MODIFIED: 'IRAY' , 'LABEL' , NNODES                           *
C  *                                                                     *
C  ***********************************************************************
C
      CALL RDATT
      CALL RDSYS
      CALL RDTTL
C
C     INITIALIZE MASTER NODE (MAKING ANY PREVIOUSLY STORED DATA
C     INACCESSABLE)
C
      CALL LSET(1,"MASTER")
      CALL ISET(1,1,0)
      CALL ISET(1,3,-1)
      CALL ISET(1,4,0)
      NNODES = 1
      CALL SPAN
C
C     SAVE THE TREE AT THIS POINT, TO INSURE AGAINST SYSTEM CRASHES
C
      CALL TSAVE
      CALL TLOAD
      RETURN
      END
```

```
CPAGE START IRAY   (ACTS AS AN ARRAY ... IM = IRAY(I,J))
CPAGE START IRAY   (ACTS AS AN ARRAY ... IM = IRAY(I,J))
      FUNCTION IRAY(I,J)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
C
C  ***********************************************************************
C  *                                                                     *
C  *  FUNCTION IRAY(I,J)                                                 *
C  *  USE: TO ACCESS THE VALUES WHICH ARE STORED IN THE                  *
C  *       PSUEDO-ARRAY 'IRAY'(I,J).  THIS ROUTINE RETURNS               *
C  *       THE VALUES FOR 'IRAY'( ,1-4) FOR THE NODE(*).                 *
C  *       SEE THE FUNCTION ARAY FOR A COMPLETE DISCUSSION.              *
C  *  CALLED BY: CALC , COPYR , DSPLAY , DSPLOT , FIND ,                 *
C  *             LISTIT , NEXT , PRENEX , PRETOT , PRNT ,                *
C  *             PRUNE , RDV , RDWT , SENSTV , SHEET ,                   *
C  *             SNSPLT                                                  *
C  *  ROUTINES CALLED: READMS(S)                                         *
C  *  VARIABLES:                                                         *
C  *       USED: I(P) , J(P) , NTAPE                                     *
C  *       MODIFIED: IRRAY(AKA ARRAY) , ISIN                             *
C  *                                                                     *
C  ***********************************************************************
C
      IF(I.NE.ISIN)CALL READMS(NTAPE,IRRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      IRAY = IRRAY(J)
      ISIN = I
      RETURN
      END
CPAGE START ISET   (SETS PSUEDO-ARRAY IRAY(I,J) = IM )
      SUBROUTINE ISET(I,J,IM)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE ISET(I,J,IM)                                            *
C  *  USE: TO SET THE VALUE OF THE PSUEDO-ARRAY 'IRAY'(I,J)              *
C  *       TO THE VALUE IM.                                              *
C  *  CALLED BY: ADD , COPYR , INITT , PRUNE , SPAN                      *
C  *  ROUTINES CALLED: READMS(S) , WRITMS(S)                             *
C  *  VARIABLES:                                                         *
C  *       USED: I(P) , IM(P) , J(P) , NTAPE                             *
C  *       MODIFIED: 'IRAY' , IRRAY(AKA ARRAY) , ISIN                    *
C  *                                                                     *
C  ***********************************************************************
C
      IF(I.NE.ISIN)CALL READMS(NTAPE,IRRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      IRRAY(J) = IM
C
C     SAVE THE NEW DATA ON DISK FILE
      CALL WRITMS(NTAPE,IRRAY,64,I,1)
```

```
      ISIN = I
      RETURN
      END
```

```
CPAGE START LABEL  (ACTS AS AN ARRAY ... "LABEL" = LABEL(I))
      FUNCTION LABEL(I)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
C
C  ***********************************************************************
C  *                                                                     *
C  *  FUNCTION LABEL(I)                                                  *
C  *  USE: TO RETURN THE LABEL ASSOCIATED WITH THE NODE(I)               *
C  *       ON THE CDC 6600, THIS LABEL CONSISTS OF TEN (10)              *
C  *       CHARACTERS OF ANY TYPE.  OTHER SYSTEMS MAY HAVE               *
C  *       SMALLER OR LARGER WORD SIZES, AND CHANGING TO                 *
C  *       THE DIFFERENT SIZE WOULD INVOLVE CHANGING THE                 *
C  *       INPUT AND OUTPUT FORMATS TO ACCOUNT FOR THE                   *
C  *       DIFFERENT NUMBER OF CHARACTERS.                               *
C  *  CALLED BY: COPYR , DSPLAY , DSPLOT , LISTIT , NUMREV ,             *
C  *             PRNT , RDV , SHEET , SPAN                               *
C  *  ROUTINES CALLED: READMS(S)                                         *
C  *  VARIABLES:                                                         *
C  *       USED:  I(P) , NTAPE                                           *
C  *       MODIFIED: IRRAY(AKA ARRAY) , ISIN , 'LABEL'                   *
C  *                                                                     *
C  ***********************************************************************
C
      IF(I.NE.ISIN)CALL READMS(NTAPE,IRRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      LABEL = IRRAY(7)
      ISIN = I
      RETURN
      END
CPAGE START LSET   (SETS PSUEDO-ARRAY LABEL(I) = "LABEL")
      SUBROUTINE LSET(I,LABELT)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE LSET(I,LABELT)                                          *
C  *  USE: TO SET THE VALUE OF THE PSUEDO-ARRAY 'LABEL' TO               *
C  *       THE VALUE LABELT.                                             *
C  *       SEE THE ROUTINES ARAY AND LABEL FOR ' COMPLETE                *
C  *       DISCUSSION.                                                   *
C  *  CALLED BY: ADD , COPYR , INITT , SPAN                              *
C  *  ROUTINES CALLED: READMS(S) , WRITMS(S)                             *
C  *  VARIABLES:                                                         *
C  *       USED: I(P) , LABELT(P) , NTAPE                                *
C  *       MODIFIED: IRRAY(AKA ARRAY) , ISIN , 'LABEL'                   *
C  *                                                                     *
C  ***********************************************************************
C
      IF(I.NE.ISIN)CALL READMS(NTAPE,IRRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      IRRAY(7) = LABELT
C     SAVE THE NEW DATA ON DISK FILE
```

```
      CALL WRITMS(NTAPE,IRRAY,64,I,1)
      ISIN = I
      RETURN
      END
```

```
CPAGE START LISTIT (PRINTS A SPECIAL FORMAT LINE)
      SUBROUTINE LISTIT
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE LISTIT                                                  *
C  *  USE: TO PRINT OUT A SPECIAL FORMAT LINE DISPLAY ...                *
C  *            (NRN VECTOR) (NODE LABEL)                                *
C  *               1 1 2 5 2   SAMPLE NOD                                *
C  *       ALTHOUGH THERE IS A STRAIGHTFORWARD FORMAT                    *
C  *       WHICH MAKES THIS UNNECESSARY, IT WAS LEFT IN                  *
C  *       FOR EASY MODIFICATION OF THAT FORMAT WHEN                     *
C  *       THIS IS IMPLEMENTED ON OTHER COMPUTER SYSTEMS.                *
C  *  CALLED BY: REASON , REVIEW , SENSTV                                *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES                                                          *
C  *       USED: 'IRAY' , 'LABEL' , LEVEL , LVL                          *
C  *       MODIFIED: I(L)                                                *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6001,LVL,(IRAY(LEVEL(I,1),1),I=1,LVL),
     2 LABEL(LEVEL(LVL,1))
6001  FORMAT(1X,=(I2,1X),A10)
      RETURN
      END
```

```
CPAGE START MENU    (DRIVER TO OPERATE NUMERIC MENU SELECTION)
      SUBROUTINE MENU(N,LIST,IWANT)
      DIMENSION LIST(N)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE MENU(N,LIST,IWANT)                                      *
C  *  USE: TO GET USER SELECTION FROM A MENU                             *
C  *       BY NUMBER.  ALTHOUGH THE ORIGINAL PROGRAM                     *
C  *       USED THIS EXTENSIVELY, THE LATER VERSIONS                     *
C  *       PRIMARILY USE IT FOR SHORT, SIMPLE SELECTIONS.                *
C  *  CALLED BY: CONTRL , PRUNE                                          *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES                                                          *
C  *       USED: N(P) , LIST(P)                                          *
C  *       MODIFIED: I(L) , IWANT(P)                                     *
C  *                                                                     *
C  ***********************************************************************
C
100   CONTINUE
      READ *,IWANT
      IF(IWANT.GE.0.AND.IWANT.LE.N)GO TO 200
      WRITE 6000
6000  FORMAT(" YOUR OPTIONS ARE ...",/,
     2 "  0)NONE OF THESE")
      WRITE 6001,((I,LIST(I)),I=1,N)
6001  FORMAT((1X,4(I2,") ",A10,2X)))
      WRITE 6002,N
6002  FORMAT(" SELECT OPTION(0-",I1,")?")
      GO TO 100
200   CONTINUE
      RETURN
      END
```

```
CPAGE START NEXT   (TRAVERSES TREE (USED AFTER PRENEX))
      SUBROUTINE NEXT
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
C
C ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE NEXT                                                 *
C  *  USE: THIS ROUTINE IS USED TO PERFORM A DEPTH-FIRST               *
C  *       SEARCH OF THE TREE, NODE BY NODE.  THE VARIABLES           *
C  *       NECESSARY ARE INITIALIZED BY PRENEX (AND PRETOT),          *
C  *       THEN THE ROUTINE NEXT WILL PERFORM THE DEPTH-FIRST         *
C  *       SEARCH, NODE BY NODE.  A NEW NODE IS ACCESSED EACH         *
C  *       TIME THAT NEXT IS CALLED.                                  *
C  *  CALLED BY: CALC , COPYR , NUMREV , SPAN , VLOAD ,               *
C  *             WLOAD , WRKSHT                                       *
C  *  ROUTINES CALLED: (NONE)                                         *
C  *  VARIABLES:                                                      *
C  *       USED: 'IRAY' , NLVLS                                       *
C  *       MODIFIED: ICONT , IDATA , IFADD , IFIND ,                  *
C  *                 INNRN , LEVEL , LVL                              *
C  *                                                                  *
C  ********************************************************************
C
C         INPUT LVL,LEVEL (CONTAINING CURRENT NODE)
C         OUTPUT  IFIND,LVL,LEVEL (OF NEXT NODE IN SEARCH)
C                 ICONT  (1 IF YES, CONTINUE WITH THIS NEW NODE;
C                         2 IF NO, SEARCH ENDS WITH INPUT NODE)
C                 IDATA  (1 IF NEW NODE IS A DATA NODE;
C                         0 IF NEW NODE IS INTERIOR NODE)
C
C  ********************************************************************
C
      ICONT = 0
      IDATA = 0
      IF(LEVEL(LVL,2).EQ.1)GO TO 200
C       THEN HAVE ALREADY GONE DOWN FROM IFIND, ELSE
C       MUST TRAVEL DOWN LINK
        IF(IRAY(LEVEL(LVL,1),2).LE.0)GO TO 200
C         THEN NO DOWN LINK EXISTS
C         ELSE TRAVEL DOWN LINK BY
C           (1) FLAGGING THAT WE ARE GOING DOWNFROM IFIND
C           (2) GETTING NEXT NODE POINTER (IRAY AND ARAY)
C           (3) CLEARING DOWNLINK INDICATOR FOR NEW NODE
C           (4) ADDING THE NEW LEVEL (LVL)
C           (5) SETTING THE IFIND POINTER TO THE NEW NODE
C           (6) SETTING YES INTO CONTINUE FLAG (ICONT)
C
C           TRAVEL DOWN LINK
            LEVEL(LVL,2) = 1
            LEVEL(LVL+1,1) = IRAY(LEVEL(LVL,1),2)
            LEVEL(LVL+1,2) = 0
            LVL = LVL + 1
            IFIND = LEVEL(LVL,1)
```

```
            ICONT = 1
            IFADD = 2
            GO TO 999
C
200   CONTINUE
C     CHECKING CROSS LINKING
      IF(LVL.EQ.1.AND.NLVLS.NE.0)GO TO 999
C       THEN DONE WITH A PARTIAL PROCESSING, ELSE
C       CONTINUE CHECKING
        IF(IRAY(IFIND,3).LE.0)GO TO 300
C         THEN NO CROSS LINK EXISTS FOR IFIND
C         ELSE CROSSLINK EXISTS, SO CROSS OVER BY
C           REPLACING CURRENT BOTTOM NODE WITH ITS
C           CROSSLINK
            LEVEL(LVL,1) = IRAY(IFIND,3)
            LEVEL(LVL,2) = 0
            IFIND = LEVEL(LVL,1)
            ICONT = 1
            IFADD = 3
            GO TO 999
300   CONTINUE
C     NO DOWN OR CROSS LINK EXISTS, BACK UP ONE LEVEL
      LVL = LVL - 1
      IF(LVL.LE.0)GO TO 999
C       THEN WE HAVE RETURNED TO TOP
      IFIND = LEVEL(LVL,1)
      GO TO 200
999   CONTINUE
      IF(IRAY(IFIND,2).LE.0)IDATA = 1
      INNRN(LVL) = IRAY(IFIND,1)
      RETURN
      END
```

```
CPAGE START NO     (FUNCTION RETURNS .TRUE. (NO) OR .FALSE. (YES))
      LOGICAL FUNCTION NO(N,QUEST)
      LOGICAL YES
      DIMENSION QUEST(1)
C
C  ***********************************************************************
C  *                                                                     *
C  *  FUNCTION NO(N,QUEST)                                               *
C  *  USE: THIS FUNCTION IS A LOGICAL FUNCTION (RETURNS                  *
C  *       .TRUE. OR .FALSE.).  IT WILL GET A Y(YES) OR                  *
C  *       N(NO) RESPONSE TO THE QUESTION STORED IN THE                  *
C  *       PASSED ARRAY QUEST.  IF THE RESPONSE IS YES,                  *
C  *       THE RETURN GIVEN IS .FALSE.                                   *
C  *       IF THE RESPONSE IS NO, THE RETURN GIVEN IS                    *
C  *       .TRUE.                                                        *
C  *       THIS ALLOWS THE PROGRAM TO BE WRITTEN SO AS                   *
C  *       TO READ MORE CLEARLY.  SEE ROUTINE YES FOR                    *
C  *       THE COUNTERPART DISCUSSION.                                   *
C  *  CALLED BY: DRIVER , NUMREV , PRUNE , REVIEW , SPAN , SUMMRY        *
C  *  ROUTINES CALLED: YES                                               *
C  *  VARIABLES:                                                         *
C  *       USED: N(P) , QUEST(P)                                         *
C  *       MODIFIED: (NONE)                                              *
C  *                                                                     *
C  ***********************************************************************
C
      NO = .TRUE.
      IF(YES(N,QUEST)) NO = .FALSE.
      RETURN
      END
```

```
CPAGE START NODIN  (GETS INPUT NRN, CALLS FIND)
      SUBROUTINE NODIN
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE NODIN                                                *
C  *  USE: THIS ROUTINE PROMPTS FOR AND READS A NODE REFERENCE        *
C  *       NUMBER (NRN) VECTOR, WHICH SPECIFIES A NODE IN THE         *
C  *       TREE (NODE MAY OR MAY NOT EXIST).  THE ROUTINE             *
C  *       FIND IS CALLED UPON TO SEARCH FOR THE INPUT                *
C  *       NODE.                                                      *
C  *  CALLED BY: ADD , COPYR , DISPLA , PRENEX , PRUNE ,              *
C  *             SENSTV , SUMMRY                                     *
C  *  ROUTINES CALLED: FIND                                           *
C  *  VARIABLES:                                                      *
C  *       USED: (NONE)                                               *
C  *       MODIFIED: ICONT , INNRN , NLVLS                            *
C  *                                                                  *
C  ********************************************************************
C
C         INPUT (NO INPUT EXCEPT MANUAL INPUT OF NRN)
C         OUTPUT NLVLS (LENGTH OF INNRN NRN VECTOR)
C                INNRN (NUMERIC NRN VECTOR)
C
C  ********************************************************************
C
      NLVLS = 0
      ICONT = 0
      WRITE 6001
6001  FORMAT(" ENTER ... NRN?")
100   CONTINUE
      NLVLS = NLVLS +1
      READ *,INNRN(NLVLS)
      IF(INNRN(NLVLS).GT.0)GO TO 100
C       THEN NOT AT END OF VECTOR
C       ELSE BACK OFF LAST ENTRY (LAST ENTRY WAS LE 0)
          NLVLS = NLVLS - 1
          IF(INNRN(1).LE.0)GO TO 999
C           THEN USER OPTED TO NOT INPUT A NRN VECTOR
C           ELSE FIND NEAREST MATCH, SUPPLYING ...
C                    IFIND,NDIFF,IFADD,LVL,LEVEL
            CALL FIND
999   CONTINUE
      RETURN
      END
```

```
CPAGE START NUMREV (REVIEW STYLE PRINT OF TREE WITH WTS AND VALUES)
      SUBROUTINE NUMREV
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SYS/NSYS,LSYS(63)
      LOGICAL YES,NO
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE NUMREV                                               *
C  *  USE: THIS ROUTINE GENERATES A REVIEW STYLE LISTING              *
C *        OF THE DATA STRUCTURE, ALONG WITH THE ASSOCIATED           *
C  *       WEIGHTS AND VALUES.                                        *
C  *  CALLED BY: DRIVER                                               *
C  *  ROUTINES CALLED: NEXT , PRENEX , PRETOT                         *
C  *  VARIABLES:                                                      *
C  *       USED: 'ARAY' , ICONT , IDATA , IFIND , INNRN , 'LABEL' ,   *
C  *             LEVEL , LVL , NDEEP , NSYS , 'VRAY'                  *
C  *       MODIFIED: I(L) , J(L) , WT(L)                              *
C  *                                                                  *
C  ********************************************************************
C
      IF(NDEEP.GT.0)GO TO 1
C       THEN PROGRAM KNOWS THE DEPTH OF THE TREE (NDEEP)
C       ELSE MUST INFORM USER THAT
          WRITE 6000
6000      FORMAT(" NUMREV ... THIS OPTION NOT ALLOWED UNTIL",
     2           " AFTER TREE HAS BEEN COLLAPSED.")
          GO TO 999
1     CONTINUE
      WRITE 6001
6001  FORMAT(" SELECT TOP NODE FOR NUMERIC REVIEW.")
      CALL PRETOT
      IF(NO(2,"REVIEW ALL NODES?"))CALL PRENEX
C       THEN GET THE TOP NODE FOR THE NUMERIC REVIEW
      IF(ICONT.EQ.0)GO TO 999
C       THEN USER OPTED OR ERRED OUT
C              ***** CREATE HEADER *****
      WRITE 6002,NDEEP,NDEEP,NDEEP,NSYS,(LSYS(J),J=1,NSYS)
6002  FORMAT(1X,=(3X),30X,"VALUES OF THE SYSTEMS",/,
     2 1X,=(3X),15X,"WEIGHT CUMWT   (USER INPUT EQUIVALENTS)",/,
     3 1X,=(3X),32X,=(A6,"-"))
C
C     ************* START THE REVIEW PRINT *************
C
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 200
C       THEN DONE SO EXIT
C       ELSE PROCESS THIS NODE AND GET THE NEXT ONE
          WT = 1.0
          IF(IDATA.EQ.0)WT = ARAY(IFIND,2)
C           THEN NEED TO CORRECT THE VALUES TO BE PRINTED
```

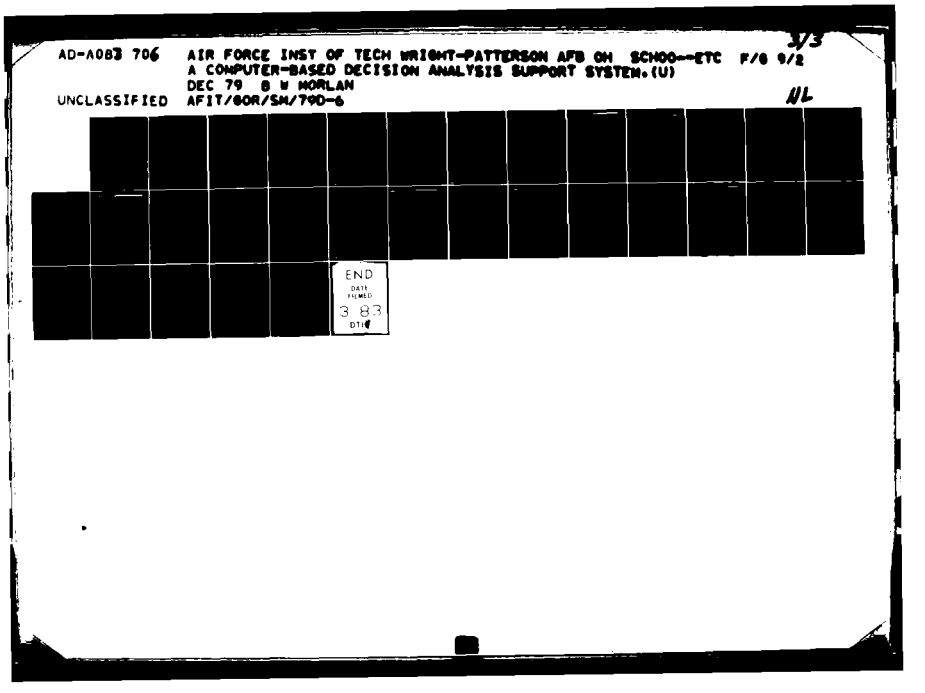
```
          ISTOP = NDEEP - LVL + 1
          WRITE 6003,LVL,(INNRN(I),I=1,LVL),LABEL(LEVEL(LVL,1)),
     2    ISTOP,ARAY(IFIND,1),ARAY(IFIND,2),
     3    ((VRAY(IFIND,J)/WT),J=1,NSYS)
6003      FORMAT(1X,=I3,1X,A10,=(3X),2F6.2,3X,10F7.2)
          CALL NEXT
      GO TO 100
C
C     ******************** EXIT **********************
C
200   CONTINUE
      WRITE 6004
6004  FORMAT(/,/)
999   CONTINUE
      RETURN
      END
```

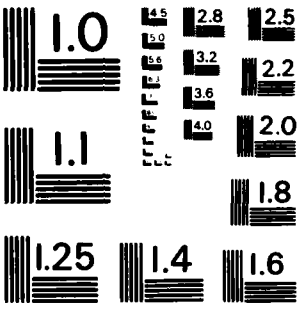
```
CPAGE START PRENEX (SETS UP TO TRAVERSE TREE (ALL OR PART) BY 'NEXT')
      SUBROUTINE PRENEX
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE PRENEX                                               *
C  *  USE: THIS ROUTINE IS THE PRE-NEXT SETUP ROUTINE.                *
C  *       IT IS CALLED BEFORE STARTING A NEXT DIRECTED               *
C  *       DEPTH-FIRST SEARCH OF THE TREE.                            *
C  *  CALLED BY: COPYR , NUMREV , REVIEW , SPAN , SUMMRY , VLOAD ,    *
C  *             WLOAD , WRKSHT                                       *
C  *  ROUTINES CALLED: NODIN , YES                                    *
C  *  VARIABLES:                                                      *
C  *       USED: 'IRAY' , NDIFF , NLVLS , NNODES                      *
C  *       MODIFIED: ICONT , IDATA , IFIND , INNRN ,                  *
C  *                 ITOTL , LEVEL , LVL                              *
C  *                                                                  *
C  ********************************************************************
C
      IF(NNODES.GT.1)GO TO 100
C       THEN THERE IS A TREE IN THE ARRAYS
C       ELSE MUST BE OPTED OUT
        WRITE 6001
6001    FORMAT(" NO TREE IN ARRAYS, CANNOT CONTINUE.")
        ICONT = 0
        GO TO 2
100   CONTINUE
      ICONT = 0
      ITOTL = 0
      CALL NODIN
      IF(NLVLS.LE.0)GO TO 2
C       THEN INPUT NODE WAS A STOP INDICATOR
      IF(NDIFF.GT.0)GO TO 2
C       THEN NODE NOT FOUND, SO SHOULD NOT CONTINUE
C       ELSE NODE WAS REQUESTED AND FOUND
        LEVEL(1,1) = IFIND
        LEVEL(1,2) = 0
        LVL = 1
1     CONTINUE
      IFIND = LEVEL(1,1)
      INNRN(1) = IRAY(IFIND,1)
      IF(IRAY(IFIND,2).LE.0)IDATA = 1
C       THEN NODE IS A DATA NODE AND FLAG AS SAME
      ICONT = 1
2     CONTINUE
      RETURN
      END
```

```
CPAGE START PRETOT (PREPARES TO TRAVERSE ENTIRE TREE)
      SUBROUTINE PRETOT
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE PRETOT                                                  *
C  *  USE: THIS ROUTINE IS USED TO SET UP FOR A NEXT                     *
C  *       DIRECTED DEPTH-FIRST SEARCH OF THE ENTIRE                     *
C  *       TREE (AS OPPOSED TO A DEPTH-FIRST SEARCH                      *
C  *       FROM A PARTICULAR NODE).                                      *
C  *  CALLED BY: CALC , NUMREV , REASON , REVIEW , SPAN , SUMMRY ,       *
C  *             VLOAD , WLOAD , WRKSHT                                 *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: 'IRAY' , NNODES                                         *
C  *       MODIFIED: ICONT , IDATA , IFIND , ITOTL ,                     *
C  *                 LEVEL , LVL , NDIFF , NLVLS                         *
C  *                                                                     *
C  ***********************************************************************
C
      ICONT = 0
      IDATA = 0
      LEVEL(1,1) = 2
      LEVEL(1,2) = 0
      LVL = 1
      NLVLS = 0
      NDIFF = 1
      IFIND = LEVEL(1,1)
      IF(IRAY(IFIND,2).LE.0)IDATA = 1
      IF(NNODES.GT.1)ICONT = 1
      INNRN(1) = IRAY(IFIND,1)
      ITOTL= 1
      RETURN
      END
```

```
CPAGE START PRNT   (PART OF DEBUGGING PACKAGE)
      SUBROUTINE PRNT
      COMMON/C/NNODES,NDEEP,TITLE(62)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE PRNT                                                    *
C  *  USE: PRINTS A GENERAL DUMP OF THE ENTIRE TREE                      *
C  *       STRUCTURE ARRAYS ('IRAY','ARAY','LABEL').                     *
C  *       IT IS CALLED ONLY WHEN THE PRINT FLAG (IPRINT)                *
C  *       HAS BEEN SET TO 1                                             *
C  *  CALLED BY: ADD                                                     *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: 'ARAY' , 'IRAY' , 'LABEL' , NNODES                      *
C  *       MODIFIED: I(L) , J(L)                                         *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6000,NNODES
6000  FORMAT(/," PRNT.  NNODES = ",I4,
     2 /,/,4X,"LABEL IRAY(*,J) J=1,4 ARAY(*,J) J=1,2")
      WRITE 6001,((LABEL(I),(IRAY(I,J),J=1,4),
     2 ARAY(I,1),ARAY(I,2)),I=1,NNODES)
6001  FORMAT(1(1X,A11,1X,4I4,2F5.3))
      WRITE 6002
6002  FORMAT(/," END OF PRNT PACKAGE PRINT.",/)
      RETURN
      END
```

```
CPAGE START PRUNE  (CLIPS OFF TREE BRANCHES)
      SUBROUTINE PRUNE
      LOGICAL YES,NO
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      DIMENSION PRMEN(5)
      DATA NPR/3/
      DATA PRMEN/"NODE+DOWN","DOWN ONLY","SOME DOWN",2*" "/
C
C  ********************************************************************
C  *                                                                  *
C  *  SUBROUTINE PRUNE                                                *
C  *  USE: THIS ROUTINE ALLOWS THE USER TO PRUNE (TO                  *
C  *       USE A KEENEY AND RAIFFA METAPHOR) THE DATA                 *
C  *       STRUCTURE.  THE USER CAN CUT A NODE PLUS                   *
C  *       ALL THE DECENDENTS, OR THE DESCENDENTS ONLY                *
C  *       (CONVERTING A BRANCHING NODE TO A DATA NODE),              *
C  *       OR JUST SELECTED DESCENDENTS.                              *
C  *  CALLED BY: DRIVER                                               *
C  *  ROUTINES CALLED: CALC , MENU , NO , NODIN , RDV ,               *
C  *                   RDWT , YES                                     *
C  *  VARIABLES:                                                      *
C  *       USED: IWANT(L) , 'LABEL' , LEVEL , NDIFF , NLVLS ,         *
C  *             NPR(L) , PRMEN(L)                                    *
C  *       MODIFIED: IDATA , IFIND , IFROM(L) , 'IRAY' ,              *
C  *                 ISAVD , LVL                                      *
C  *                                                                  *
C  ********************************************************************
C         THERE ARE THREE TYPES OF PRUNE
C
C     1 A
C     1 1 AA
C     1 1 1 AAA  BECOMES 1 A      OR  1 A     OR  1 A
C     1 1 2 AAB                      1 1 AA      1 1 AA
C                                                1 1 2 AAB
C
C   ********************************************************************
C
C
100   CONTINUE
C
C     GET NODE
C     EXIT IF (USER OPTS OUT OR (NODE IS NOT FOUND
C              AND USER OPTS NOT TO TRY AGAIN))
C
C
      WRITE 6001
6001  FORMAT(" ENTER NODE TO BE PRUNED.")
      CALL NODIN
      IF(NLVLS.LE.0)GO TO 999
      IF(NDIFF.EQ.0)GO TO 200
C       THEN EVERYTHING IS SET FOR PRUNING
C       ELSE
```


AD-A083 706
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A COMPUTER-BASED DECISION ANALYSIS SUPPORT SYSTEM.(U)
DEC 79 B W MORLAN
UNCLASSIFIED AFIT/GOR/SM/79D-6
3/3
NL
END
DATE FILMED
3 83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

```
          IF(NO(3,"NODE NOT FOUND, TRY AGAIN?"))GO TO 999
          GO TO 100
C
200   CONTINUE
      IFROM = 2
C
C     SELECT PRUNING MODE (1-3)
C     EXIT IF (USER OPTS OUT (MODE 0))
C
      WRITE 6002
6002  FORMAT(" SELECT ... TYPE OF PRUNING OPERATION?")
      CALL MENU(NPR,PRMEN,IWANT)
      IF(IWANT.EQ.0)GO TO 999
      ISAVD = 0
      GO TO (1,2,3),IWANT
C
C     ******************** PRUNE MODE 1 ********************
C
C     THE FOLLOWING SECTION PRUNES THE SELECTED
C     NODE FROM THE TREE COMPLETELY
C
1     CONTINUE
      LVL = LVL - 1
      IFROM = IRAY(IFIND,4)
      IF(IRAY(IFROM,3).EQ.IFIND)GO TO 15
C       THEN NODE WAS A CROSSLINK TO NODE (FROM)
C       ELSE IT WAS A DOWNLINK
          CALL ISET(IFROM,2,IRAY(IFIND,3))
          IF(IRAY(IFROM,2).GT.0)CALL ISET(IRAY(IFROM,2),4,IFROM)
          GO TO 400
15    CONTINUE
      CALL ISET(IFROM,3,IRAY(IFIND,3))
      IF(IRAY(IFROM,3).GT.0)CALL ISET(IRAY(IFROM,3),4,IFROM)
      GO TO 400
C
C     ******************** PRUNE MODE 2 ********************
C
C     THE FOLLOWING SECTION PRUNES ALL THE DOWNLINK
C     BRANCHES FROM THE SELECTED NODE
C
2     CONTINUE
      CALL ISET(IFIND,2,-1)
      GO TO 400
C
C     ******************** PRUNE MODE 3 ********************
C
C     THE FOLLOWING SECTION ALLOWS SELECTIVE
C     PRUNING OF THE DOWNLINK BRANCHES FROM
C     THE SELECTED NODE.
C
3     CONTINUE
      IFIND = IRAY(IFIND,IFROM)
      IF(IFIND.LE.0)GO TO 400
      WRITE 6003,IRAY(IFIND,1),LABEL(IFIND)
```

```
6003  FORMAT(" CURRENT DESCENDENT NODE IS ",I3," (",A10,").")
      IF(YES(2,"CUT THIS DESCENDENT?"))GO TO 300
      IFROM = 3
      GO TO 3
300   CONTINUE
      CALL ISET(IRAY(IFIND,4),IFROM,IRAY(IFIND,3))
      IF(IRAY(IFIND,3).LE.0)GO TO 400
      CALL ISET(IRAY(IFIND,3),4,IRAY(IFIND,4))
      IFROM = 3
      GO TO 3
C
C     ********** CORRECT WEIGHTS OR VALUES (AS NECESSARY ) **********
C
C     THE FOLLOWING SECTION DETERMINES WHETHER PRUNING
C     CHANGED: (1) THE NUMBER OF DOWNLINK BRANCHES TO A NODE
C     OR (2) CREATED A NEW DATA NODE.  THEN THE
C     APPROPRIATE ACTION IS TAKEN: (1) GET NEW WEIGHTS (RDWT)
C     OR (2) GET NEW DATA VALUES (RDV).  THEN (RE)CALCULATE
C     THE TREE VALUES IF NECESSARY AND REQUESTED
C
400   CONTINUE
      IFIND = LEVEL(LVL,1)
      IDATA = 0
      IF(IRAY(IFIND,2).LE.0)IDATA = 1
C       THEN BOTTOM NODE IS NOW A DATA NODE
      IF(IDATA.EQ.0)CALL RDWT
C       THEN NEED TO ADJUST WEIGHTS
C       ELSE NEED TO ENTER NEW VALUES
      IF(IDATA.EQ.1)CALL RDV
C
      GO TO 100
C       TO GET THE NEXT NODE TO BE PRUNED
C
999   CONTINUE
      IF(ISAVD.EQ.0.AND.
     2 YES(2,"RECALCULATE TREE?"))CALL CALC
      RETURN
      END
```

```
CPAGE START RDATT  (READS IN ATTRIBUTE CHARACTERISTIC (REGRET/VALUE))
      SUBROUTINE RDATT
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE RDATT                                                    *
C  *  USE: THIS ROUTINE ALLOWS THE USER TO SELECT THE                     *
C  *       TYPE OF VALUES WHICH ARE TO BE ASSIGNED TO                     *
C  *       THE SYSTEMS AT THE DATA NODES.  PRIMARILY,                     *
C  *       THE CHOICES ARE VALUES OR REGRETS/DEFICIENCIES.                *
C  *       THIS IS USED IN PROMPTING THE USER FOR DATA                    *
C  *       NODE VALUES.                                                   *
C  *  CALLED BY: DRIVER , INITT                                           *
C  *  ROUTINES CALLED: (NONE)                                             *
C  *  VARIABLES:                                                          *
C  *       USED: (NONE)                                                   *
C  *       MODIFIED: ISAVD , LATT                                         *
C  *                                                                      *
C  ************************************************************************
C
      WRITE 6001
6001  FORMAT(" ENTER ATTRIBUTE (REGRET OR VALUE)?")
      READ 5001,LATT
5001  FORMAT(A10)
      ISAVD = 0
      RETURN
      END
```

```
CPAGE START RDSYS  (READS IN LABELS FOR THE SYSTEMS)
      SUBROUTINE RDSYS
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/SYS/NSYS,LSYS(63)
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE RDSYS                                                    *
C  *  USE: THIS ROUTINE ALLOWS THE USER TO ENTER THE                      *
C  *       LABELS TO BE USED IN REFERRING TO THE SYSTEMS                  *
C  *       WHICH ARE BEING EVALUATED.                                     *
C  *  CALLED BY: DRIVER , INITT                                           *
C  *  ROUTINES CALLED: (NONE)                                             *
C  *  VARIABLES:                                                          *
C  *       USED: (NONE)                                                   *
C  *       MODIFIED: ISAVD , LSYS , NSYS                                  *
C  *                                                                      *
C  ************************************************************************
C
1     CONTINUE
      NSYS = 1
100   CONTINUE
      WRITE 6001,NSYS
6001  FORMAT(" ENTER ... SYSTEM ",I2," LABEL?")
      READ 5001,LSYS(NSYS)
5001  FORMAT(A10)
      IF(LSYS(NSYS) EQ." ".OR.
     2   LSYS(NSYS).EQ."DONE")GO TO 200
      NSYS = NSYS + 1
      GO TO 100
200   CONTINUE
      NSYS = NSYS - 1
      IF(NSYS.GE.1)GO TO 300
      WRITE 6002
6002  FORMAT(" SORRY ... YOU MUST ENTER AT LEAST ONE SYSTEM.")
      GO TO 1
300   CONTINUE
      ISAVD = 0
      RETURN
      END
```

```
CPAGE START RDTTL  (READS THE TITLE FOR THE DATA STRUCTURE)
      SUBROUTINE RDTTL
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE RDTTL                                                   *
C  *  USE: THIS ROUTINE ALLOW THE USER TO ENTER A TITLE                  *
C  *       TO BE USED FOR THE CURRENT DATA STRUCTURE.                    *
C  *  CALLED BY: DRIVER , INITT                                          *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: (NONE)                                                  *
C  *       MODIFIED: 'CRAY'                                              *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6001
6001  FORMAT(" ENTER A TITLE FOR THIS DATA STRUCTURE.")
      CALL CSET(1)
      RETURN
      END
```

```
CPAGE START RDV    (READS IN VALUES FOR ONE NODE)
      SUBROUTINE RDV
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SYS/NSYS,LSYS(63)
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE RDV                                                     *
C  *  USE: THIS READS THE VALUES WHICH ARE ASSOCIATED WITH               *
C  *       THE SELECTED DATA NODE.  THE CALLING ROUTINE CHECKS           *
C  *       THAT THE NODE IS A DATA NODE (IDATA = 1).                     *
C  *  CALLED BY: WLOAD                                                   *
C  *  ROUTINES CALLED: EOF(S)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: IFIND , LATT , LEVEL , LSYS , LVL , NSYS                *
C  *       MODIFIED: I(L) , ITOTL , J(L) , V(L)                          *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6001,LVL,(IRAY(LEVEL(I,1),1),I=1,LVL),
     2 (LABEL(LEVEL(J,1)),J=1,LVL)
6001  FORMAT(/," GIVEN THE FOLLOWING ... ",=(I2,1X),
     2 10(A10,"~"))
      WRITE 6002,LATT,(LSYS(J),J=1,NSYS)
6002  FORMAT(" ENTER THE MEASURE OF ",A10,
     2 " FOR EACH SYSTEM",/,
     3       " IN THE FOLLOWING ORDER ...",/,1X,
     4       10(A5,"~"))
      WRITE 6003
6003  FORMAT(1X,"?")
      DO 100 J = 1 , NSYS
      READ *,V
      IF(EOF(5LINPUT).EQ.0.0)GO TO 50
C       THEN USER IS STILL INPUTTING DATA
C       ELSE USER HAS OPTED OUT OF RDV AND VLOAD
          ITOTL = 0
          GO TO 999
50    CONTINUE
      CALL VSET(IFIND,J,V)
100   CONTINUE
      WRITE 6004
6004  FORMAT(" ENTER COMMENTS ON THESE VALUES.")
      CALL CSET(IFIND)
999   CONTINUE
      RETURN
      END
```

```
CPAGE START RDWT   (READS WEIGHTS FOR DESCENDENT NODES)
      SUBROUTINE RDWT
      LOGICAL YES,NO
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      DIMENSION  AIN(20)
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE RDWT                                                     *
C  *  USE: THIS ROUTINE READS AND NORMALIZES THE WEIGHTS                  *
C  *       WHICH APPLY TO THE DOWNLINK BRANCHES TO THE                    *
C  *       SELECTED NODE.  THE CALLING ROUTINE CHECKS THAT                *
C  *       THE NODE IS NOT A DATA NODE (IDATA = 0).                       *
C  *  CALLED BY: VLOAD                                                    *
C  *  ROUTINES CALLED: DSPLAY , DVIDE , EOF(S) , NO , SUM                 *
C  *  VARIABLES:                                                          *
C  *       USED: LEVEL , LVL                                              *
C  *       MODIFIED: AIN(L) , 'ARAY' , 'CRAY' , I(L) ,                    *
C  *                 IFIND , IFINDT(L) , ITOTL , NCROSS(L) ,              *
C  *                 ZNORM(L)                                             *
C  *                                                                      *
C  ************************************************************************
C
C         INPUT  LVL,LEVEL (INDICATING THE NODE BEING INPUT)
C
      CALL DSPLAY
      IFIND = LEVEL(LVL,1)
      IFINDT = IRAY(IFIND,2)
C
C     COUNTING THE NUMBER OF BRANCHES TO BE WEIGHTED
C
      NCROSS = 1
100   CONTINUE
      IFINDT = IRAY(IFINDT,3)
      IF(IFINDT.LE.0)GO TO 200
C       THEN NO MORE DESCENDENTS
C       ELSE CONTINUE ACROSS
          NCROSS = NCROSS + 1
          GO TO 100
C
200   CONTINUE
C     PRINT OLD WEIGHTS
      WRITE 6001,NCROSS,(I,I=1,NCROSS)
6001  FORMAT(" FACTOR      ",=("(",I2,") "))
C
400   CONTINUE
C
C     THE FOLLOWING SECTION ELICITS THE NEW WEIGHTS
C     WEIGHTS ARE INPUT AS RATIOS, THEN NORMALIZED
C     AND ECHO PRINTED FOR VERIFICATION BY USER
C
      WRITE 6002
```

```
6002  FORMAT(" NEWWEIGHTS?")
      READ *,(AIN(I),I=1,NCROSS)
      IF(EOF(5LINPUT).EQ.0.0)GO TO 450
C       THEN ALL DATA WAS INPUT
C       ELSE USER IS EXITTING FROM RDWT AND WLOAD
          ITOTL = 0
          GO TO 999
450   CONTINUE
      IF(NCROSS.EQ.1)CALL ASET(IRAY(IFIND,2),1,0.0)
C       THEN THERE IS ONLY ONE DOWN BRANCH
C            SO SET WEIGHT TO 0, UNLESS OVERRIDDEN
      ZNORM = SUM(NCROSS,AIN)
      IF(ZNORM.LT.0.0)GO TO 999
C       THEN USER OPTED OUT
C       ELSE
          IF(ZNORM.GT.0.0)CALL DVIDE(NCROSS,AIN,ZNORM)
C           THEN NORMALIZE THE INPUT VECTOR
          WRITE 6003,((IFIX(100.0*AIN(I)+0.5)),I=1,NCROSS)
6003      FORMAT(" NORMALIZED ",15I5)
          IF(NO(3,"ARE THESE WEIGHTS OKAY?"))GO TO 400
C           THEN WEIGHTS ARE NOT OKAY, GET THEM AGAIN
C           ELSE MAKE THE INPUT WEIGHTS PERMANENT
500   CONTINUE
C
C     SAVE THE NORMALIZED, VERIFIED WEIGHTS IN THE
C     APPROPRIATE 'ARAY' LOCATIONS.
C
      IFINDT = IRAY(IFIND,2)
      DO 510 I = 1 , NCROSS
        CALL ASET(IFINDT,1,AIN(I))
        IFINDT = IRAY(IFINDT,3)
510   CONTINUE
C
      WRITE 6004
6004  FORMAT(" ENTER COMMENTS ON THESE WEIGHTS.")
      CALL CSET(IFIND)
999   CONTINUE
      RETURN
      END
```

```
CPAGE START REASON (RETURNS THE RATIONALE FOR WEIGHTS OR VALUES)
      SUBROUTINE REASON
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE REASON                                                   *
C  *  USE: THIS ROUTINE WRITES THE RATIONALES WHICH WERE                  *
C  *       COLLECTED WITH THE WEIGHTS OR VALUES INPUT BY                  *
C  *       THE USER.                                                      *
C  *  CALLED BY: DRIVER                                                   *
C  *  ROUTINES CALLED: LISTIT , NEXT , PRENEX                             *
C  *  VARIABLES:                                                          *
C  *       USED: 'CRAY' , ICONT , IFIND , ITOTL                           *
C  *       MODIFIED: I(L)                                                 *
C  *                                                                      *
C  ************************************************************************
C
      WRITE 6001
6001  FORMAT(" REASON ... RETRIEVING RATIONALES STORED WHEN ",
     2       "INPUTTING WEIGHTS AND VALUES.")
C
      CALL PRETOT
      IF(YES(4,"GET THE RATIONALES FROM ALL THE NODES?"))GO TO 100
C       THEN GETTING THEM ALL
C       ELSE GET THEM ONE AT A TIME
C
1     CONTINUE
      CALL PRENEX
C
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 999
C       THEN NO TREE IN ARRAYS, OR DONE TRAVERSING
C       ELSE PROCESS THIS NODE
          CALL LISTIT
          CALL CRAY(IFIND)
          IF(ITOTL.EQ.0)GO TO 1
C           THEN ONLY DOING ONE NODE AT A TIME
C           ELSE GET NEXT NODE
              CALL NEXT
              GO TO 100
999   CONTINUE
      RETURN
      END
```

```
CPAGE START REVIEW (PRINTS A REVIEW OF THE TREE)
      SUBROUTINE REVIEW
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE REVIEW                                                   *
C  *  USE: GENERATES A LISTING OF ALL DESCENDENT NODES                    *
C  *       FROM A SELECTED NODE, USING THE NEXT-DIRECTED                  *
C  *       DEPTH-FIRST SEARCH.                                            *
C  *  CALLED BY: DRIVER                                                   *
C  *  ROUTINES CALLED: LISTIT , NEXT , PRENEX , PRETOT                    *
C  *  VARIABLES:                                                          *
C  *       USED: ICONT                                                    *
C  *       MODIFIED: (NONE)                                               *
C  *                                                                      *
C  ************************************************************************
C
1     CONTINUE
      WRITE 6000
6000  FORMAT(" SELECT TOP NODE FOR REVIEW.")
C
      CALL PRETOT
      IF(NO(2,"REVIEW ALL NODES?"))CALL PRENEX
C       THEN USER WANTS TO GO FROM A SELECTED NODE
C       ELSE USE THE PRESET ASSUMPTION OF PRETOT
C
      WRITE 6001
6001  FORMAT(/)
C
C     START THE REVIEW PRINT
C
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 200
C       THEN DONE SO EXIT
C       ELSE PROCESS THIS NODE AND GET THE NEXT ONE
          CALL LISTIT
          CALL NEXT
          GO TO 100
C
200   CONTINUE
      WRITE 6001
999   CONTINUE
      RETURN
      END
```

```
CPAGE START SENSTV (WEIGHT SENSITIVITY ANALYSIS)
      SUBROUTINE SENSTV
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SEN/WTMIN,WTMAX,WTDLT
      COMMON/SYS/NSYS,LSYS(63)
      DIMENSION DUM(20)
      DATA BLANK/1H / , STAR/1H*/
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE SENSTV                                                  *
C  *  USE: THIS ROUTINE GENERATES A SENSITIVITY ANALYSIS                 *
C  *       ON THE WEIGHT OF ONE BRANCH.  THE WEIGHTS                     *
C  *       USED ARE CUMULATIVE WEIGHTS, AND CAN BE                       *
C  *       ALLOWED (SELECTED BY USER) TO RUN AVER ANY                    *
C  *       SUBINTERVAL OF THE CLOSED INTERVAL (0,1).                     *
C  *       VALUES ARE TESTED AND PLOTTED FOR THE ENDPOINTS               *
C  *       PLUS NINE (9) INTERNAL, EVENLY SPACED POINTS.                 *
C  *  CALLED BY: DRIVER                                                  *
C  *  ROUTINES CALLED: LISTIT , NODIN , SNSPLT                           *
C  *  VARIABLES:                                                         *
C  *       USED: 'ARAY' , IFIND , 'IRAY' , LEVEL , LSYS ,                *
C  *             NLVLS , NNODES , NSYS                                   *
C  *       MODIFIED: DUM(L) , I(L) , IFINDT(L) ,                         *
C  *                 IWORK(L) , J(L) , JMAX(L) ,                         *
C  *                 VHOLD(L) , 'VRAY' , WT(L) , WTDLT , WTMAX ,         *
C  *                 WTMIN , WTNOW(L)                                    *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6001
6001  FORMAT(" SENSITIVITY ANALYSES FOLLOW.")
1     CONTINUE
      WRITE 6002
6002  FORMAT(" BRANCH FOR WHICH WEIGHT (CUMWT) IS TO BE PERTURBED.")
      CALL NODIN
      IF(NLVLS.LE.0.OR.NDIFF.GT.0)GO TO 999
C     THEN USER OPTED OR ERRED OUT
      CALL LISTIT
      WRITE 6003,ARAY(IFIND,2)
6003  FORMAT(" CURRENT CUMULATIVE WEIGHT IS ",F4.2,
     2       " MINIMUM CUMWT (0-1) IS?")
      READ *,WTMIN
      IF(WTMIN.LT.0.0)GO TO 999
      WRITE 6004
6004  FORMAT(" MAXIMUM CUMWT (0-1) IS?")
      READ *,WTMAX
      IF(WTMAX.LT.WTMIN.OR.WTMAX.GT.1.0)GO TO 999
      WTDLT = (WTMAX-WTMIN)/10.0
      IF(WTMIN.EQ.WTMAX)WTMAX = WTMIN - 0.1
C       THEN DO JUST ONE SAMPLE, WITH WT=WTMIN
```

```
C       ELSE WILL DO ELEVEN SAMPLES
      IFINDT = LEVEL(1,1)
      IWORK = NNODES + 1
      WTNOW = WTMIN
      WT = 1.0
      IF(IRAY(IFIND,2).GE.1)WT = ARAY(IFIND,2)
      WRITE 6005,(LSYS(I),I=1,NSYS)
6005  FORMAT("    WEIGHT ",11(1X,A5,2X))
100   CONTINUE
      JMAX = 1
      DO 200 J = 1 , NSYS
      VHOLD =
     2 (VRAY(IFINDT,J)-VRAY(IFIND,J)*ARAY(IFIND,2)/WT)
     3 *(1.0-WTNOW)/(1.0-ARAY(IFIND,2))
     4 +VRAY(IFIND,J)*WTNOW/WT
      CALL VSET(IWORK,J,VHOLD)
      IF(VRAY(IWORK,J).GT.VRAY(IWORK,JMAX))JMAX = J
      DUM(J) = BLANK
200   CONTINUE
      DUM(JMAX) = STAR
      WRITE 6006,WTNOW,((VRAY(IWORK,J),DUM(J)),J=1,NSYS)
6006  FORMAT(1X,F7.2,1X,11(F7.2,A1))
      WTNOW = WTNOW + WTDLT
      IF(WTNOW.LE.WTMAX)GO TO 100
C       THEN CONTINUE WITH THIS SENSITIVITY ANALYSIS
C       ELSE START NEXT SENSITIVITY ANALYSIS
          CALL SNSPLT
          GO TO 1
999   CONTINUE
      RETURN
      END
```

```
CPAGE START SHEET  (SINGLE LINE OF WRKSHT OUTPUT)
      SUBROUTINE SHEET
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SYS/NSYS,LSYS(63)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE SHEET                                                   *
C  *  USE: GENERATES ONE NODE'S WORTH OF WORKSHEET.                      *
C  *  CALLED BY: WRKSHT                                                  *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: IDATA , IFIND , 'IRAY' , 'LABEL' , LATT ,               *
C  *             LEVEL , LSYS , LVL , NSYS                               *
C  *       MODIFIED: I(L) , IFINDT(L) , J(L)                             *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6001,LVL,(IRAY(LEVEL(I,1),1),I=1,LVL),
     2 LABEL(LEVEL(LVL,1))
6001  FORMAT(1X,=(I2,1X),A10)
      IF(IDATA.EQ.1)GO TO 200
C       THEN IS A DATA NODE
C       ELSE IS A BRANCHING NODE
      IFINDT = IRAY(IFIND,2)
100   CONTINUE
      IF(IFINDT.LE.0)GO TO 400
      WRITE 6002,LVL,(" ",I=1,LVL),IRAY(IFINDT,1),LABEL(IFINDT)
6002  FORMAT(1X,=A3,I2,1X,A10,"  (WT:____)")
      IFINDT = IRAY(IFINDT,3)
      GO TO 100
200   CONTINUE
      WRITE 6003,"SYSTEMS = ",(LSYS(I),I=1,NSYS)
      WRITE 6003,LATT,("  (____)",J=1,NSYS)
6003  FORMAT(/,1X,10(A10,"-"),4(/,1X,10(A10,"-")))
400   CONTINUE
      WRITE 6004
6004  FORMAT(/)
      RETURN
      END
```

```
CPAGE START SNSPLT (WEIGHT SENSITIVITY ANALYSIS PLOT)
      SUBROUTINE SNSPLT
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      COMMON/SEN/WTMIN,WTMAX,WTDLT
      COMMON/SYS/NSYS,LSYS(63)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE SNSPLT                                                  *
C  *  USE: PRODUCES A PLOT OF THE SENSITIVITIES PERFORMED                *
C  *       BY THE ROUTINE SENSTV.  AS IN THE CASE OF THE                 *
C  *       PLOTS OF THE VALUES (DSPLOT), ALL PLOTTING                    *
C  *       IS DONE USING FORTRAN FORMAT STATEMENTS, AND                  *
C  *       THE RESOLUTION IS +/- 2 UNITS.                                *
C  *       THE RESOLUTION IS +/- 2 UNITS.                                *
C  *  CALLED BY: SENSTV                                                  *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: ALFA(L) , 'ARAY' , BLANK(L) , IFIND , 'IRAY' ,          *
C  *             LEVEL , NNODES , NSYS , RSCALE(L) , STAR(L) ,           *
C  *             VERT(L) , 'VRAY' , WTDLT , WTMAX , WTMIN                *
C  *       MODIFIED: DSPL(L) , I(L) , IFINDT(L) , IPUT(L) ,              *
C  *                 J(L) , VHOLD(L) , WT(L) , WTNOW(L)                  *
C  *                                                                     *
C  ***********************************************************************
C
      DIMENSION ALFA(20),DSPL(51)
      DATA ALFA/1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,
     2          1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT/
      DATA RSCALE/2.0/
      DATA BLANK/1H / , STAR/1H*/
      DATA VERT/1H+/
      IFINDT = LEVEL(1,1)
      WRITE 6001
6001  FORMAT(/,/)
      WRITE 6002
6002  FORMAT(10X,
     2 "0--------20--------40--------60--------80-------100")
      WTNOW = WTMIN
      WT = 1.0
      IF(IRAY(IFIND,2).GE.1)WT = ARAY(IFIND,2)
100   CONTINUE
      DO 200 I = 2 , 50
      DSPL(I) = BLANK
200   CONTINUE
      DO 300 I = 1 , 51 , 10
      DSPL(I) = VERT
300   CONTINUE
      DO 400 J = 1 , NSYS
      VHOLD =
     2 (VRAY(IFINDT,J)-VRAY(IFIND,J)*ARAY(IFIND,2)/WT)
```

```
     3 *(1.0-WTNOW)/(1.0-ARAY(IFIND,2))
     4 +VRAY(IFIND,J)*WTNOW/WT
       IPUT = VHOLD/RSCALE + 1.4999
       DSPL(IPUT) = ALFA(J)
400    CONTINUE
       WRITE 6003,WTNOW,DSPL
6003   FORMAT(1X,"WT =",F4.2,1X,51A1)
       WTNOW = WTNOW + WTDLT
       IF(WTNOW.LE.WTMAX)GO TO 100
C        THEN CONTINUE WITH THIS SENSITIVITY ANALYSIS
C        ELSE START NEXT SENSITIVITY ANALYSIS
999    CONTINUE
       WRITE 6002
       WRITE 6001
       RETURN
       END
```

```
CPAGE START SPAN   (ADDS NODES BY SPANNING SETS)
      SUBROUTINE SPAN
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/LEVEL/NLVLS,INNRN(20),IFIND,NDIFF,IFADD,LVL,LEVEL(20,3)
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ***************************************************************
C  *                                                             *
C  *  SUBROUTINE SPAN                                            *
C  *  USE: ALLOWS USER TO ENTER THE BRANCHES FROM A NODE         *
C  *       BY LABEL ONLY.  THE ROUTINE GENERATES THE             *
C  *       NEW NRN VECTOR AUTOMATICALLY (FIRST NODE IS           *
C  *       1, SECOND IS 2, ETC.).  THE ROUTINE THEN              *
C  *       AUTOMATICALLY 'WALKS' DOWN THE NEW NODES,             *
C  *       ALLOWING THE USER TO ADD BRANCHES BELOW THOSE         *
C  *       NEW NODES.  THE USER CAN END THE CURRENT SET          *
C  *       OF BRANCHES BY ENTERING A NULL STRING OR A            *
C  *       STRING 'DONE', OR CAN EXIT THE ROUTINE ALTOGETHER     *
C  *       BY ENTERING A STRING 'EXIT'.                          *
C  *  CALLED BY: DRIVER , INITT                                  *
C  *  ROUTINES CALLED: NEXT , NO , PRENEX , PRETOT               *
C  *  VARIABLES:                                                 *
C  *       USED: (NONE)                                          *
C  *       MODIFIED: 'ARAY' , I(L) , ICONT , IFADD ,             *
C  *                 IFIND , IFINDT(L) , INNRN , 'IRAY' ,        *
C  *                 'LABEL' , LABELT(L) , LEVEL , LVL ,         *
C  *                 NNRN(L) , NLVLS , NNODES                    *
C  *                                                             *
C  ***************************************************************
C
      WRITE 6001
6001  FORMAT(" SELECT TOP NODE FOR SPANNING TREE BUILDING.")
C
      CALL PRETOT
      IF(NO(2,"SPAN AT ALL NODES?"))CALL PRENEX
C       THEN USER WANTS TO SPAN FROM A SELECTED NODE
C       ELSE USE THE PRESET ASSUMPTION OF PRETOT
C
      IFADD = 2
      IF(ICONT.EQ.1)GO TO 100
C       THEN ALL SET TO ADD NODES
C       ELSE
          IF(NO(4,"DID YOU WANT TO BUILD A NEW TREE?"))GO TO 400
C           THEN EXIT
C           ELSE MUST RESET FOR A NEW TREE
              LEVEL(1,1) = 1
              LEVEL(1,2) = 1
              INNRN(1) = 0
              LVL = 1
              IFIND = LEVEL(1,1)
              IFADD = 3
              ICONT = 1
              NLVLS = 0
```

```
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 400
C       THEN DONE
C       ELSE START TO ADD DOWNLINKS TO THIS NODE
      NNRN = 0
      IFINDT = IFIND
      WRITE 6002,LVL,(INNRN(I),I=1,LVL),LABEL(LEVEL(LVL,1))
6002  FORMAT(" ADDING DOWNLINKS TO NODE ...",/,
     2 =I3,1X,A10)
200   CONTINUE
      WRITE 6003
6003  FORMAT(" LABEL?",1X)
      READ 5001,LABELT
5001  FORMAT(A10)
      IF(LABELT.EQ."EXIT")GO TO 400
C       THEN USER IS EXITTING
C       ELSE CONTINUE WITH PROCESSING
          IF(LABELT.EQ." ".OR.LABELT.EQ."DONE")GO TO 300
C           THEN DONE WITH THIS NODE
C           ELSE CONTINUE WITH THIS NODE
              NNRN = NNRN + 1
              NNODES = NNODES + 1
              CALL ISET(IFINDT,IFADD,NNODES)
              CALL ISET(NNODES,1,NNRN)
              CALL ISET(NNODES,2,-1)
              CALL ISET(NNODES,3,-1)
              CALL ISET(NNODES,4,IFINDT)
              CALL ASET(NNODES,1,0.0)
              CALL ASET(NNODES,2,0.0)
              CALL LSET(NNODES,LABELT)
              IFADD = 3
              IFINDT = NNODES
              GO TO 200
300   CONTINUE
      CALL NEXT
      IFADD = 2
      GO TO 100
400   CONTINUE
      RETURN
      END
```

```
CPAGE START SUM    (SUMS ELEMENTS OF A VECTOR)
      FUNCTION SUM(N,A)
      DIMENSION A(N)
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE SUM                                                      *
C  *  FUNCTION SUM(N,A)                                                   *
C  *  USE: RETURNS THE SUM OF THE ELEMENTS OF A VECTOR                    *
C  *       (A) OF LENGTH N                                                *
C  *  CALLED BY: RDWT                                                     *
C  *  ROUTINES CALLED: (NONE)                                             *
C  *  VARIABLES:                                                          *
C  *       USED: A(P) , N(P)                                              *
C  *       MODIFIED: I(L) , SUM(L)                                        *
C  *                                                                      *
C  ************************************************************************
C
      SUM = 0.0
      DO 100 I = 1 , N
      SUM = SUM + A(I)
100   CONTINUE
      RETURN
      END
```

```
CPAGE START SUMMRY (DRIVER FOR MASS DSPLAY)
      SUBROUTINE SUMMRY
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE SUMMRY                                                   *
C  *  USE: GENERATES A DSPLAY AND DSPLOT OF A SELECTED                    *
C  *       NODE PLUS ALL THE DESCENDENTS OF THAT NODE.                    *
C  *       USER CAN SELECT TO GET THE INFORMATION FOR                     *
C  *       THE ENTIRE DATA STRUCTURE.                                     *
C  *  CALLED BY: DRIVER                                                   *
C  *  ROUTINES CALLED: DSPLAY , DSPLOT , NEXT , NO ,                      *
C  *                   PRENEX , PRETOT                                    *
C  *  VARIABLES:                                                          *
C  *       USED: 'CRAY' , ICONT , IDATA                                   *
C  *       MODIFIED  (NONE)                                               *
C  *                                                                      *
C  ************************************************************************
C
      WRITE 6001
6001  FORMAT(" SUMMARY FOLLOWS.")
C
      CALL PRETOT
      IF(NO(3,"SUMMARIZE ALL THE NODES?"))CALL PRENEX
C       THEN SUMMARIZE FROM A SELECTED NODE
C       ELSE USE THE PRESET ASSUMPTION OF PRETOT
C
      WRITE 6002
6002  FORMAT("1")
C
C     PRINT TITLE
      CALL CRAY(1)
C
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 999
      IF(IDATA.EQ.1)GO TO 200
C       THEN NODE IS A DATA NODE AND CANNOT BE DISPLAYED
C       ELSE DISPLAY AND PLOT THIS NODE
          CALL DSPLAY
          CALL DSPLOT
200   CONTINUE
      CALL NEXT
      GO TO 100
C
999   CONTINUE
      RETURN
      END
```

```
CPAGE START TAPER  (ROUTINE DRIVES TREE FILES)
      SUBROUTINE TAPER
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
C
C  ***********************************************************************
C  *                                                                     *
C  *                                                                     *
C  *  SUBROUTINE TAPER                                                   *
C  *  USE: THIS ROUTINE MANIPULATES THE DATA FILES.                      *
C  *       IT ALLOWS THE USER TO PROCESS UP TO THREE (3)                 *
C  *       INDEPENDENT DATA STRUCTURES WITHOUT 'CROSSTALK'.              *
C  *       THE LIMIT OF THREE IS NOT A HARD LIMIT, ADDING                *
C  *       FURTHER FILES IS A SIMPLE MATTER OF ALLOWING                  *
C  *       FOR MORE IN THE PROGRAM HEADER AND IN THE                     *
C  *       CHECK PERFORMED IN TAPER (1.LE.NTAPE.LE.3).                   *
C  *  CALLED BY:DRIVER                                                   *
C  *  ROUTINES CALLED: INITT , TLOAD , TSAVE                             *
C  *  VARIABLES:                                                         *
C  *       USED: NNODES                                                  *
C  *       MODIFIED:  NTAPE                                              *
C  *                                                                     *
C  ***********************************************************************
C
      WRITE 6001,NTAPE
6001  FORMAT(" THE CURRENT TREE IS NUMBER ",I2,".")
      CALL TSAVE
100   CONTINUE
      WRITE 6002
6002  FORMAT(" WITH WHICH TREE WOULD YOU LIKE TO WORK?")
      READ *,NTAPE
      IF(NTAPE.GE.1.AND.NTAPE.LE.3)GO TO 200
C       THEN INPUT TAPE NUMBER WAS GOOD
C       ELSE INFORMATIVE MSG AND GET AGAIN
        WRITE 6003,NTAPE
6003    FORMAT(" INPUT TREE NUMBER OF ",I2," IS NOT ALLOWED.")
        GO TO 100
200   CONTINUE
      CALL TLOAD
      IF(NNODES.GT.0)GO TO 300
C       THEN TLOAD OPENED A FILE WITH A TREE STRUCTURE
C       ELSE MIGHT AS WELL SAVE USER ONE INPUT, AND GO TO OPTION 'NEW'
        WRITE 6004
6004    FORMAT(" FILE HAS NO TREE, PROGRAM GOING TO OPTION 'NEW'.")
        CALL INITT
300   CONTINUE
      RETURN
      END
```

```
CPAGE START TLOAD  (TO OPEN MASS STORAGE)
      SUBROUTINE TLOAD
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
      LOGICAL YES,NO
      COMMON/SYS/NSYS,LSYS(63)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE TLOAD                                                   *
C  *  USE: THIS ROUTINE OPENS THE NTAPE SELECTED DATA FILE               *
C  *       THEN INITIALIZES THE DIRECT ACCESS ARRAYS IF                  *
C  *       NECESSARY, OR READS IN THE TITLE, ATTRIBUTE,                  *
C  *       AND SYSTEM ARRAYS FROM THE DIRECT ACCESS FILE.                *
C  *       IF THERE IS NO DATA IN THE FILE, IT WILL SELECT               *
C  *       THE 'NEW' OPTION FOR THE USER.                                *
C  *  CALLED BY: DRIVER , TAPER                                          *
C  *  ROUTINES CALLED: OPENMS(S) , READMS(S) , WRITMS(S)                 *
C  *  VARIABLES:                                                         *
C  *       USED: NTAPE                                                   *
C  *       MODIFIED: ATTDUM , I(L) , IRRAY(AKA ARRAY) , ISIN ,           *
C  *                 ISTOP(L) , LATT , LSYS , MASTER , NDEEP ,           *
C  *                 NLOUD , NNODES , NSYS , TITLE                       *
C  *                                                                     *
C  ***********************************************************************
C
C     MAIN SIZER NOW AT 500 NODES
      NLOUD = 505
C     MAIN SIZER
      WRITE 6001,NTAPE
6001  FORMAT(" NOW OPENING TREE FILE NUMBER ",I2,".")
      ISIN = 0
      CALL OPENMS(NTAPE,MASTER,NLOUD,0)
      IF(MASTER(2).NE.0)GO TO 3
C       THEN TAPE HAS ALREADY BEEN OPENED ONCE
C       ELSE MUST INITIALIZE THE DATA LOCATIONS
      DO 1 I = 1 , 64
      IRRAY(I) = 0
1     CONTINUE
      ISTOP = NLOUD - 1
      DO 2 I = 1 , ISTOP
      CALL WRITMS(NTAPE,IRRAY,64,I,0)
2     CONTINUE
      NNODES = 0
      GO TO 4
3     CONTINUE
      I = NLOUD - 4
      CALL READMS(NTAPE,NNODES,64,I)
      I = NLOUD - 3
      CALL READMS(NTAPE,LATT,64,I)
      I = NLOUD - 2
      CALL READMS(NTAPE,NSYS,64,I)
```

```
4       CONTINUE
        RETURN
        END
```

```
CPAGE START TSAVE  (TO CLOSE MASS STORAGE)
      SUBROUTINE TSAVE
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,IRRAY(64),NLOUD,MASTER(505)
      COMMON/SYS/NSYS,LSYS(63)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE TSAVE                                                   *
C  *  USE: THIS ROUTINE CLOSES THE CURRENT DATA FILE.                    *
C  *       IN DOING SO, IT WRITES THE TITLE, ATTRIBUTE,                  *
C  *       AND SYSTEM LABELS INTO THE DATA FILE.                         *
C  *  CALLED BY: DRIVER , TAPER                                          *
C  *  ROUTINES CALLED: CLOSMS(S) , WRITMS(S)                             *
C  *  VARIABLES:                                                         *
C  *       USED: LATT , LSYS , NDEEP , NLOUD , NNODES , NSYS ,           *
C  *             NTAPE , TITLE                                         *
C  *       MODIFIED: I(L) , ISAVD                                        *
C  *                                                                     *
C  ***********************************************************************
C
      I = NLOUD - 4
      CALL WRITMS(NTAPE,NNODES,64,I,1)
      I = NLOUD - 3
      CALL WRITMS(NTAPE,LATT,64,I,1)
      I = NLOUD - 2
      CALL WRITMS(NTAPE,NSYS,64,I,1)
      CALL CLOSMS(NTAPE)
      WRITE 6001,NTAPE
6001  FORMAT(" JUST CLOSED (SAVING) TREE FILE ",I2,".")
      ISAVD = 1
      RETURN
      END
```

```
CPAGE START VLOAD  (SUBDRIVER TO GET VALUES FOR DATA NODES)
      SUBROUTINE VLOAD
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ***************************************************************************
C  *                                                                         *
C  *  SUBROUTINE VLOAD                                                       *
C  *  USE: TO DRIVE THE ENTRY OF THE DATA NODE VALUES FOR                    *
C  *       THE DATA STRUCTURE.  THE USER MAY OPT TO ENTER                    *
C  *       A SINGLE NODE AT A TIME (SELECTED BY NRN VECTOR),                 *
C  *       OR FOR A NEXT-DIRECTED DEPTH-FIRST SEARCH FOR                     *
C  *       DATA NODES, WITH DATA ENTRY AT EACH DATA NODE.                    *
C  *  CALLED BY: WVLOAD                                                      *
C  *  ROUTINES CALLED: NEXT , PRENEX , PRETOT , PRETOT , RDV                 *
C  *  VARIABLES:                                                             *
C  *       USED: ICONT , IDATA , ITOTL                                       *
C  *       MODIFIED: (NONE)                                                  *
C  *                                                                         *
C  ***************************************************************************
C
      WRITE 6001
6001  FORMAT(" YOU ARE ABOUT TO BE ASKED FOR ",
     2       " DATA NODE VALUES.")
C
      CALL PRETOT
      IF(YES(3,"GET VALUES FOR ALL DATA NODES?"))GO TO 100
C       THEN PROGRAM WAS ALL SET TO GO
C       ELSE GET EACH NODE TO BE EVALUATED SEPERATELY
C
1     CONTINUE
      CALL PRENEX
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 999
      IF(IDATA.EQ.1)CALL RDV
      IF(ITOTL.EQ.0)GO TO 1
C       THEN NOT DOING ENTIRE TREE, SO GET NEXT NRN
          CALL NEXT
          GO TO 100
999   CONTINUE
      RETURN
      END
```

```
CPAGE START VRAY   (ACTS AS AN ARRAY ... V = VRAY(I,J))
      FUNCTION VRAY(I,J)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
C
C  ***********************************************************************
C  *                                                                     *
C  *  FUNCTION VRAY(I,J)                                                 *
C  *  USE: RETURNS THE VALUE OF THE PSUEDO-ARRAY                         *
C  *       'VRAY'(I,J).  'VRAY'(I,J) IS THE INPUT OR                     *
C  *       CALCULATED VALUE FOR THE I-TH NODE FOR SYSTEM J.              *
C  *  CALLED BY: CALC , DSPLAY , DSPLOT , NUMREV , SENSTV ,              *
C  *             SNSPLT                                                  *
C  *  ROUTINES CALLED: READMS(S)                                         *
C  *  VARIABLES:                                                         *
C  *       USED: I(P) , J(P) , NTAPE                                     *
C  *       MODIFIED: ARRAY , IPULL(L) , ISIN                             *
C  *                                                                     *
C  ***********************************************************************
C
      IPULL = 7 + J
      IF(I.NE.ISIN)CALL READMS(NTAPE,ARRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
      VRAY = ARRAY(IPULL)
      ISIN = I
      RETURN
      END
CPAGE START VSET   (SETS PSUEDO-ARRAY VRAY(I,J)=V )
      SUBROUTINE VSET(I,J,V)
      COMMON/ATT/LATT,ATTDUM(63)
      COMMON/C/NNODES,NDEEP,TITLE(62)
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/RRAY/ISIN,ARRAY(64),NLOUD,MASTER(505)
      COMMON/SYS/NSYS,LSYS(63)
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE VSET(I,J,V)                                             *
C  *  USE: USED TO SET THE VALUE OF THE PSUEDO-ARRAY                     *
C  *       'VRAY'(I,J) EQUAL TO V.                                       *
C  *       SEE FUNCTION ARAY AND SUBROUTINE ASET FOR                     *
C  *       FURTHER DISCUSSION.                                           *
C  *  CALLED BY: CALC , RDV , SENSTV                                     *
C  *  ROUTINES CALLED: READMS(S) , WRITMS(S)                             *
C  *  VARIABLES:                                                         *
C  *       USED: I(P) , J(P) , NTAPE , V(P)                              *
C  *       MODIFIED: ARRAY , IPUT(L) , ISIN , 'VRAY'                     *
C  *                                                                     *
C  ***********************************************************************
C
      IPUT = 7 + J
      IF(I.NE.ISIN)CALL READMS(NTAPE,ARRAY,64,I)
C       THEN WRONG DATA CELL IS IN CORE
```

```
      ARRAY(IPUT) = V
C
C     SAVE THE NEW DATA ON DISK FILE
      CALL WRITMS(NTAPE,ARRAY,64,I,1)
      ISIN = I
      RETURN
      END
```

```
CPAGE START WLOAD  (DRIVER TO LOAD WEIGHTS FOR NODES)
      SUBROUTINE WLOAD
      COMMON/CNTRL/IPRINT,ISAVD,NTAPE,MODE
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE WLOAD                                                    *
C  *  USE: DRIVES THE ENTRY OF WEIGHTS FOR BRANCHED NODES.                *
C  *       LIKE VLOAD, IT CAN BE USED TO ENTER ONE NODE                   *
C  *       AT A TIME, OR CAN BE USED IN THE NEXT-DIRECTED                 *
C  *       SEARCH MODE, WHERE EACH BRANCHING NODE IS                      *
C  *       VISITED FOR ENTRY OF WEIGHTS.                                  *
C  *  CALLED BY: WVLOAD                                                   *
C  *  ROUTINES CALLED: NEXT , PRENEX , PRETOT , RDWT , YES                *
C  *  VARIABLES:                                                          *
C  *       USED: ICONT , IDATA , ITOTL                                    *
C  *       MODIFIED: (NONE)                                               *
C  *                                                                      *
C  ************************************************************************
C
      WRITE 6001
6001  FORMAT(" YOU ARE ABOUT TO ENTER WEIGHTS FOR NODES.")
C
      CALL PRETOT
      IF(YES(3,"WEIGHT ALL BRANCHING NODES?"))GO TO 100
C       THEN PROGRAM WAS ALL SET TO GO
C       ELSE GET EACH NODE TO BE WEIGHTED SEPERATELY
C
1     CONTINUE
C
C     SELECT NODE OR ALL AND PREPARE TO ENTER WEIGHTS
C
      CALL PRENEX
C
100   CONTINUE
      IF(ICONT.EQ.0)GO TO 999
C       THEN NO TREE OR DONE TRAVERSING
      IF(IDATA.EQ.0)CALL RDWT
C       THEN NOT A DATA NODE, SO WEIGHTS MAKE SENSE
      IF(ITOTL.EQ.0)GO TO 1
C       THEN NOT DOING ENTIRE TREE, SO GET NEXT NRN
C       ELSE GET NEXT NODE AND CONTINUE
          CALL NEXT
          GO TO 100
999   CONTINUE
      RETURN
      END
```

```
CPAGE START WRKSHT (GENERATES WORKSHEET)
      SUBROUTINE WRKSHT
      COMMON/NEX/ICONT,IDATA,ITOTL
      LOGICAL YES,NO
C
C  ************************************************************************
C  *                                                                      *
C  *  SUBROUTINE WRKSHT                                                   *
C  *  USE: GENERATES A WORKSHEET OF A NODE AND ITS                        *
C  *       DESCENDENTS, OR OF THE ENTIRE DATA STRUCTURE.                  *
C  *       THIS WORKSHEET CAN THEN BE USED BY THE USER                    *
C  *       TO WRITE DOWN THE WEIGHTS AND VALUES WHICH                     *
C  *       ARE ASSOCIATED WITH THE DATA STRUCTURE.                        *
C  *  CALLED BY: DRIVER                                                   *
C  *  ROUTINES CALLED: NEXT , PRENEX , PRETOT , SHEET , YES               *
C  *  VARIABLES:                                                          *
C  *       USED: ICONT                                                    *
C  *       MODIFIED: (NONE)                                               *
C  *                                                                      *
C  ************************************************************************
C
      WRITE 6001
6001  FORMAT(" WORKSHEET GENERATION FOLLOWS.")
C
      CALL PRETOT
      IF(YES(5,"DO YOU WANT A WORKSHEET FOR THE ENTIRE TREE?"))GO TO 200
C       THEN PROGRAM WAS READY TO GO
C       ELSE GET THE TOP NODE TO BE USED
      CALL PRENEX
C
C     WRITE OUT THE WORKSHEET
C
200   CONTINUE
      IF(ICONT.EQ.0)GO TO 999
C       THEN DONE WITH TREE
C       ELSE WRITE OUT THIS NODE AND GET NEXT NODE
          CALL SHEET
          CALL NEXT
          GO TO 200
999   CONTINUE
      WRITE 6002
6002  FORMAT(/,/)
      RETURN
      END
```

```
CPAGE START WVLOAD (DRIVES WLOAD, VLOAD AND CALC)
      SUBROUTINE WVLOAD
      LOGICAL YES,NO
C
C  ***********************************************************************
C  *                                                                     *
C  *  SUBROUTINE WVLOAD                                                  *
C  *  USE: DRIVES THE USE OF THE ROUTINES WLOAD , VLOAD ,                *
C  *       AND CALC.  THESE THREE WERE GROUPED UNDER A                   *
C  *       SEPERATE SUBDRIVER BECAUSE THEY ARE STRONGLY                  *
C  *       RELATED.                                                      *
C  *  CALLED BY: DRIVER                                                  *
C  *  ROUTINES CALLED: CALC , VLOAD , WLOAD , YES                        *
C  *  VARIABLES:                                                         *
C  *       USED: (NONE)                                                  *
C  *       MODIFIED: (NONE)                                              *
C  *                                                                     *
C  ***********************************************************************
C
      IF(YES(2,"WEIGHT NODES?"))CALL WLOAD
      IF(YES(3,"INPUT DATA NODE VALUES?"))CALL VLOAD
      IF(YES(2,"(RE)CALCULATE TREE?"))CALL CALC
      RETURN
      END
```

```
CPAGE START YES    (FUNCTION RETURNS .TRUE. (YES) OR .FALSE. (NO))
      LOGICAL FUNCTION YES(N,QUEST)
      DIMENSION QUEST(1)
C
C  ***********************************************************************
C  *                                                                     *
C  *  FUNCTION YES(N,QUEST)                                              *
C  *  USE: THIS FUNCTION WAS SET UP TO MAKE THE REST                     *
C  *       OF THE PROGRAM READ MORE CLEARLY.  IT WILL                    *
C  *       RETURN THE LOGICAL VALUE OF .TRUE. IF THE USER                *
C  *       RESPONDS WITH A Y(YES) TO THE QUESTION WHICH                  *
C  *       WAS PASSED IN THE ARRAY QUEST.  IT RETURNS THE                *
C  *       LOGICAL VALUE .FALSE. IF THE RESPONSE IS N(NO).               *
C  *       ANY OTHER RESPONSE RESULTS IN AN INFORMATIVE                  *
C  *       MESSAGE, AND A REPEAT OF THE QUESTION.                        *
C  *       THIS IS PRIMARILY DONE AS A MEANS OF MAKING                   *
C  *       THE SOURCE CODE READ CLEANER.                                 *
C  *  CALLED BY: NO , PRUNE , REASON , VLOAD ,                           *
C  *             WLOAD , WRKSHT , WVLOAD                                 *
C  *  ROUTINES CALLED: (NONE)                                            *
C  *  VARIABLES:                                                         *
C  *       USED: N(P) , QUEST(P)                                         *
C  *       MODIFIED: I(L) , YEANAY(L)                                    *
C  *                                                                     *
C  ***********************************************************************
C
C     ******************** PROGRAM FLOW ********************
C
C     PROMPT USER WITH THE INPUT QUESTION; READ RESPONSE (*);
C     IF (THE RESPONSE DOES NOT START WITH A 'Y' OR AN 'N')
C       THEN (REMIND USER THAT THE ONLY RECOGNIZABLE INPUTS
C             ARE THOSE WHICH START WITH A 'Y' OR AN 'N';
C             AND START OVER)
C       ELSE (ASSUME THAT THE RESPONSE WAS 'N' (FALSE);
C             IF (RESPONSE WAS 'T')
C                 THEN (SET RETURN TO TRUE))
C     EXIT
C
1     CONTINUE
      WRITE 6001,(QUEST(I),I=1,N)
6001  FORMAT(1X,6A10)
      READ 5001,YEANAY
5001  FORMAT(A1)
      YES = .FALSE.
      IF(YEANAY.EQ."Y".OR.YEANAY.EQ."N") GO TO 2
C       THEN YEANAY IS A LEGAL ANSWER
      WRITE 6002
6002  FORMAT(" I ONLY UNDERSTAND Y(YES) OR N(NO).")
      GO TO 1
2     CONTINUE
      IF(YEANAY.EQ."Y") YES = .TRUE.
      RETURN
      END
```

VITA

Bruce W. Morlan was born and raised in southern Minnesota. He received a Bachelor of Science from Michigan State University in Mathematics in June 1979. He received his Air Force commission through ROTC, and served from December 1973 until June 1975 in the MINUTEMAN Missile Combat Crew Force at Malmstrom Air Force Base, Montana. He then served from June 1975 until his entry into the School of Engineering, Air Force Institute of Technology as a Trajectory Applications Analyst, in the 544 ARTW/Trajectory Division, Offutt Air Force Base, Nebraska.

END

DATE
FILMED

3-83

DTIC